Enactive Robotics: An Action-State Model for Concurrent Machine Control

Enactive Robotics

An Action-State Model for Concurrent Machine Control

by

Jose Luis García del Castillo y López

Bachelor of Architecture, Universidad de Sevilla, 2007
Master in Architectural Innovation: Technology and Design (prtl.), Universidad de Sevilla, 2010
Master in Design Studies: Technology, Harvard University, 2013

Submitted to the Graduate School of Design
in partial fulfillment of the requirements for the degree of

Doctor of Design

at

Harvard University

in Cambridge, MA, United States of America

February 2019

Jose Luis García del Castillo y López
Enactive Robotics: An Action-State Model for Concurrent Machine Control
February 2019

Thesis Supervisor: Prof. Martin Bechthold.
Thesis Advisors: Prof. Panagiotis Michalatos and Prof. Stefanie Mueller.

Cite (MLA 8th):
García del Castillo, Jose L. *Enactive Robotics: An Action-State Model for Concurrent Machine Control*. 2019. Harvard University, D.Des. Dissertation.

Published online at the Digital Access to Scholarship at Harvard repository:

A recording of the defense lecture can be found on https://youtu.be/DYnQM-8BYBY

Abstract

Industrial robots have been around since the 70s, with massive rates of adoption in the manufacturing industry. Additionally, during the last decade, there has been an increasing interest in the potential of robotic making in non-engineering fields, such as digital fabrication, architecture and art installations, with designers, researchers and artists experimenting with creative applications of these technologies. However, the typical tools used to program and control robots usually fail to address the needs of these groups.

Most robots can only be controlled by writing routines through their own graphical user interfaces or vendor-specific programming languages, which often require significant knowledge of spatial transformations, forward and inverse kinematics, mechanical engineering and computer science. These requirements make robots notoriously hard to program, and pose a great entry barrier, especially for novice and non-technical users. Moreover, and similarly to 3D printers, robot programming tools are biased towards the offline control style, one where all the planning and decision making are pre-generated on a digital environment and, upon execution of the compiled instruction file, the programmer becomes completely detached from it. This model is suitable for highly calibrated and predictable environments, but can hardly accommodate more complex forms of control such as responding to feedback from the context, adapting to changing conditions on a construction site or on-the-fly decision making by a controller agent.

This research introduces *Enactive Robotics*, a conceptual model for the design of concurrent control systems for mechanical actuators. The main goal of this model is to blur the distinction between creating and executing a robotic program, integrating them into a process where behavior can be *enacted* on the machine during the design phase. Drawing inspiration from developmental and cognitive theories, the model is grounded on the capacity of a central decision-making agent to interface in real time with the control system via a set of high-level, universal and platform-agnostic requests named *actions*. These actions conform the atomic units of cognitive interaction with the robot, and their effect on a particular device is dependent on its nature and state. This paradigm crucially involves considering the large-scale shift between mechanical and computational run times, and proposes the centrality of a *state* representation as the core mediator between them. The action-

state model seeks to break from the unidirectional offline control paradigm, and favor programming styles that are reactive to changes in the dynamic execution of the robot, rather than prescriptive about it.

The main thesis in this dissertation is that applications built following the principles of the Enactive Robotics model provide an easier and more immediate entry point to robotics for novel users, since they provide an enactive, rather than symbolic representation of the system, hence aiding the cognitive processes that lead to understanding motion planning and control. Additionally, it provides a framework with greater depth of possibilities for advanced users, in which its real-time nature and immediate feedback facilitates experimentation, flow of thought and creative inquiry. While the work presented in this dissertation focuses mainly on industrial robotic arms, it will be shown how this model can be extended to any programmable machine that performs spatial motion.

In this dissertation the general architecture of the model is presented, as well as two sample technical implementations following these principles. The first implementation is a pure .NET library designed for power-users and tech-savvy individuals, while the second is an ecosystem of UI-based applications and utility libraries geared towards novice and entry-level users. A collection of projects built with these implementations is presented as case studies, to showcase the capacity of the model to systematically enable richer interaction paradigms with robotic systems. Furthermore, the results of a controlled user study are presented, in order to evidence the capacity of the model to provide an easier and more accessible entry point to robot programming for novice users.

Acknowledgements

I owe immense gratitude to my advisor Prof. Martin Bechthold for the inspiration, guidance and mentorship in this dissertation; if there is any scholarship to be found in this document, he is definitely the cause. I would also like to thank Prof. Panagiotis Michalatos and Prof. Stefanie Mueller for their advice throughout this research, and for encouraging me to ask the right questions and speculate on even better answers.

This work was not a one man show, not even close. It is difficult to acknowledge all the people who have been part of this journey, from courses to workshops, from projects to papers... the Enactive Robotics family has grown large in the last two years, and all of you have left your imprint in the many forms this project has taken. I thank each and every one of the creative minds I have worked with, have provided feedback to the work in this dissertation, and had the patience to deal with the very early stages of the Robot Ex Machina project. A good starting point for this massive handshake are all the individuals listed on Appendix A: Case Studies Data. A special shout out goes to Nono, for being part of the journey from its very beginning, to Hakim, for being one of the earliest and deepest adopters of the technologies described in this work, and to team Sul, Hyeonji and Francis, for all the hard work and dedication they have contributed.

I would also like to thank all the participants in the Controlled User Study for volunteering to this experiment and their valuable feedback, and to Prof. Andrew Witt for facilitating it.

A special mention goes to my colleagues and friends from the Material Processes and Systems Group at the GSD. They have provided the best infrastructural, moral and daily support that any doctoral researcher could wish for. Great things lay ahead of us.

I would like to very specially acknowledge Autodesk Inc. for their support to the work behind this dissertation. Had it not been for the inspiring work of my colleagues at the Generative Design Group, the help and dedication of the BUILD Space team, and the leadership of Matt Jezyk, this dissertation would not have been possible. Thank you for believing in this project.

Similarly, I owe a great deal of gratitude for the support received from the Schodek family, the Real Colegio Complutense at Harvard, the office of the Dean and the DDes program. It is through their mission that the ideas behind my research could materialize into this dissertation, and may one day become part of our daily lives.

A personal round of gratitude goes to the Broaklanders; the entertainment provided was critical to the success of this quest! Very specially, I would like to thank Paggy for her love, patience and unmatched "gif" support; I would not have made this without you.

I would like to thank my family for their unconditional support at all times, including the most recent members Ana, Estanislao and Victoria, and our new forthcoming one. The work in this dissertation is for them, and for the future they will live in.

And finally, this dissertation is dedicated with my deepest love to my grandfather Rafael, who taught me the value of fixing things, rather than replacing them; I wish you were here to share it with you.

Contents

List of Figures

1 Introduction

Robots are coming.

During the second half of the 20th century, and the beginning of the 21st, robots have permeated into many aspects of our daily lives. Robot automation has increasingly been adopted in every industry at which labor-intensive repetitive processes could be substituted by machines. This has been particularly the case in the field of manufacturing, where mass production of identical objects would capitalize on the precision and reliability of industrial robotic arms. Today, any modern good that we may consume has been manipulated by some kind of automated machine: cars, furniture, food, electronics, etc. Virtually all objects that surround us have been touched by robots.

In the last decade, robot adoption has kicked off in non-industrial contexts too. An expanding interest in the potential of robotics has grown into fields such as digital fabrication, architecture and art installations, with designers, researchers and artists experimenting with creative applications of these technologies. More recently, we are witnessing an increasing presence of these machines in our domestic environments: from vacuum cleaners and voice-controlled gadgets in our homes, to assistive devices for impaired individuals, and very soon, delivery drones and autonomous vehicles. Robots seem to be following a similar path than that of computers: from large and expensive machines only accessible to researchers, military and industry, to smaller personal versions to assist us in our daily duties.

But we are not there yet.

Robots are becoming faster, cheaper, smaller and "softer" —safer for humans to be around. But just like it happened with computers, the challenge for mass adoption is not a purely technical one: it involves millions of human minds as well. And while hardware is becoming increasingly better, very fast, I would like to argue that the tools we use to control these devices —the software running on them—, have barely changed during the past half century.

1.1 Robots, Experts, and the Rest of Us

Robots are difficult to program.

One of the first challenges is the significant lack of universality in robot control systems. Within industry, robot manufacturers usually provide highly proprietary environments that require the use of custom programming languages, and it is fairly common for vendors to offer their own authored tools for programming and simulating their products. For instance, ABB Robotics offers their licensed RobotStudio (2018) platform to program robot cells in the RAPID language (2013); KUKA provides KUKA.Sim to simulate programs written in Kuka Robot Language (2003); and Universal Robots uses the URScript language (2018) to drive 6-axis robotic arms through URSim. Different robot brands requiring different programming languages is the first entry barrier to robotics, tying the initial learning process to the products of a particular company.

The kinematic similarities between six-axis robotic arms, and the universal nature of spatial dynamics, make it possible to abstract some of the principles behind motion planning into high-level interfaces, which can in turn be post-processed into device-specific languages. This is the core principle behind third-party robot programming environments such as a RoboDK (2018) or Visual Components (2018) that try to unify this lack of standards. However, these tools are closed source tools, with usage often restricted to the acquisition of expensive licenses, becoming an additional entry barrier for novice users to the field of robotics.

Knowledge of the principles that govern machine control is the second great barrier newcomers face when attempting to delve into robotics. Most robot programming environments require considerable domain expertise from the user. These systems are designed by and for engineers, developers, integrators and implementers, and assume their audience has an elevated degree of familiarity with the theoretical and applied foundations of robotics: spatial transformations, forward and inverse kinematics, robot mechanics, computer programming, regulatory systems, etc. The Robot Operating System ROS (Quigley et al. 2009) is notorious for the depth of possibilities it allows, its modularity, openness and extensibility, yet at the same time, it is dreaded for its complexity, steep learning curve and difficulty to set up. As a consequence, its main adopters are a reduced community of power users formed by academics, research labs and highly trained individuals.

Figure 2-1: These are MIT architecture students using a 6-axis robot arm to cut complex surfaces into the foam block on the right. To create the toolpaths, they first designed the surfaces and Rhino, then used RhinoCAM to create instructions for the robot arm. The cutting end is an electric hot knife. To regulate the temperature of the hot knife, the student standing on the left is observing the cutting and flipping the switch on a surge protector on and off. This is an extremely intelligent controller for an extremely simple control task. However, despite these students being proficient at technologies such as CAD, CAM, and robot control, the system integration is tedious enough that the easiest way for them to get their system running was to incorporate an imprecise yet very expensive controller–a human.

Figure 1 - MIT architecture students using an industrial 6-axis robotic arm to cut foam blocks with a hot knife. "System integration is tedious enough" that a human supervisor was situated outside the system, controlling the temperature. Image from Peek 2016.

Peek discusses how "robotic arms are notoriously tedious to program" (2016, p. 34), and illustrates it with an example: a group of students trying to use a six-axis robot arm with a hot knife to cut shapes out of a foam block. Given the complexity of the task and the difficulty in integrating the control mechanisms, the group needed to resort to having a human in the loop manually controlling the temperature switch of the knife.

> "This is an extremely intelligent controller for an extremely simple control task. However, despite these students being proficient at technologies such as CAD, CAM and robot control, the system integration is tedious enough that the easiest way for them to get their system running was to incorporate an imprecise yet very expensive controller—a human." (Peek 2016, p. 38)

There is an abundance of highly specialized robot programming tools for highly skilled individuals. However, without the proper training and time investment, robots are hardly accessible to more general audiences. For the most part, non-expert audiences are interested in getting the job done, without caring so much yet about the whys and the hows. It is unrealistic to assume that most people will go through formal training in order to programs robots; it is probably abusive to demand that they should.

The creation and deployment of websites used to be a proprietary domain of the tech-savvy, but nowadays, a wide variety of tools, interfaces and platforms have opened up this new basic need to a much wider audience. Just like it happened with computers, a future with pervasive, programmable and customizable robots for the masses may involve the creation of systems that bypass the need for extensive training, and provide an accessible entry point in more immediate, human terms.

1.2 The Control Bias, or the 3D Printing Paradigm

Robot programming is heavily biased.

The industrial origin of robots made early designers of their programming environments favor a very particular control style: the offline programming model. In this model, all the motion planning, instructions and logic are pre-defined inside some form of digital programming framework, become post-processed into a program in the device's native language, loaded separately to the controller and executed offline, without a live connection to the tool from which it was generated. This is for instance the typical use case for any common 3D printing operation.

Such paradigm is well suited for highly calibrated and predictable environments, where precise assumptions about their state can be done a priori. However, it is particularly constraining for systems with elevated degrees of uncertainty, or where robot behavior is mainly driven in response to evolving conditions around it. This is especially the case of robotic systems based on sensory input, environmental feedback, material properties, interactive installations, human-robot collaboration, etc. Decision mechanisms have to be created to account for all possible permutations of a changing context, and as the number and complexity of its factors rises, the design of these control systems grows exponentially impossible.

Many authors have identified the limitations of this model, especially in the context of humans making things with machines. Landay (2009) discusses how the next frontier in design tools for personal fabrication is making the modeling and simulation phases *concurrent*. Bechthold (2010) analyses the attempts at robotic construction by the turn of the 20th century, and advocates for models where *robots complement* conventional construction, rather than substitute it. Willis et al. (2011) propose *interactive fabrication* as the new paradigm that seeks to break with the unidirectionality and detachment of current fabrication methods, and bring the crafter back in direct touch with the fabricated output. Mueller (2016) argues that the offline interaction model with digital fabrication machines is not suitable for non-technical users, and explores different levels of interaction units on *personal fabrication* systems, enabling direct manipulation and continuous feedback. Ultimately, most authors agree that hardware development has reached a rather mature state, and that the big challenge to push the boundaries of human-machine making lies on the design of software interfaces to mediate this interaction.

The offline bias could be partially blamed for the "poor" robotic fabrication example previously described in Peek (2016). Integrating robot motion and knife temperature was so tedious that the students opted for controlling the switch manually. And, as the control mechanism requires all actions to be previously defined before program execution, the human controller is unable to synchronize his manual operation with significant stages of program execution; they need to "feel" when to turn the switch on or off. Pre-canned programs hardly leave room for any real-time intervention, on-the-fly decision-making or "human integration." The consequence is that the programs' outcomes are strictly determined by the original master plan; as designers of those programs, we can only cross our fingers and hope the result will be faithful to our blind vision. Or try again.

1.3 Enactive Learning

The offline control paradigm in itself represents a significant entry barrier, considered this time from a cognitive point of view.

As this model favors a programming style where all the planning, decision-making and outcomes must be predetermined before program execution, it requires from the programmer substantial domain expertise to successfully envision and implement a task. But assuming an individual is new to robots, and is willing to invest all the time and effort necessary to learn the skills to program them, how can they acquire this expertise? How can knowledge be gained in order to program a system whose prerequisite is that very same knowledge? It feels like a chicken or egg conundrum.

Too often, the response to this question is abstract training. A typical robot curriculum may start by describing the mechanical configuration of a robot, breaking down the algebra behind forward and inverse kinematics, analyzing the dynamics of motion, fundamentals of computer programming, etc., all of it characterized through geometry, math and physics at a theoretical level. I once took an engineering class on fundamentals of robotics, and a month had passed before I was able to write my first robot task on MATLAB. My training was comprehensive and exhaustive, and the nerd in me could not have been any happier than the day I wrote my first inverse kinematics solver. Yet, while this model is suited for individuals seeking formal and long-lasting training in robotics, it becomes too much of an overhead for all of those who just want to get the job done. No matter how much I loved that class, I am probably not the target audience of this dissertation.

I have taught robotics myself to too many students to know firsthand how frustrating formal training can be for a newcomer; how tedious it can be to put together and run your first program, especially for a platform so tangible and immediate as a physical robot. But I think there is hope. And I say so because, on every one of those workshops, and for every single student I have taught robotics to, I have been able to witness their deep and sincere smile of accomplishment that very moment when they make the robot move for the first time; it is a magical moment, it is invaluable.

I strongly believe that the problem lies on the symbolic nature of this learning process. Newcomers are typically introduced to robotics through abstract reasoning, backed by further abstract constructs such as mathematics or computer science. Symbolic learning favors those well versed in the conceptual tools that support it, but is rather detached from the tangibility and immediacy of the robot as an "object-to-think-with" (Papert 1980). But moreover, it is in stark contrast with the way we typically learn about the world around us.

Early pioneers of human-computer interaction (HCI) founded their research on the wealth of cognitive theories of the 20th century to design their own interaction models. Piaget's *constructivist theory* (Gruber and Vonèche 1977) had a tremendous influence in authors arguing for a more "learning through making" approach, such as Papert's *constructionism* in the LOGO project (1980). And among the most influential theories would be Bruner's research on developmental psychology (1966). Bruner would argue that cognition takes place

sequentially over three stages, defined by what he called *enactive*, *iconic* and *symbolic representations*. In his model, children shifted from the concrete to the abstract by first learning through action, allowing them to generate and understand images as their proxies, which would ultimately be elevated into language representations of those concepts. Bruner's model would become the foundation of modern proposals for HCI paradigms such as Shneiderman's *direct manipulation* as an alternative to traditional programming languages (1983), or the *programming by example* ideals by D. Smith et al. (2000). Ultimately, Bruner's model would be the inspiration for Kay's famous slogan "doing with images makes symbols" (1990).

Robots are a platform arguably less abstract and more tangible than a computer, which lend themselves much better to the immediacy of direct manipulation and action-based learning. However, offline control detaches the user from the robot and biases the understanding of how to program them to the symbolic domain. It makes learning a much harder enterprise and wastes an incredible affordance in the medium. Learning robotics should start on the robot itself, rather than from the math behind it, and robot programming tools should favor learning styles that elevate users into symbolic reasoning through the vehicle of enactive participation.

1.4 Creative Inquiry

The recent democratization of digital fabrication tools has sparked renewed interest in the field of robotics from more creative communities. The increasing availability of 3D printers, laser cutters or general Computer Numeric Control (CNC) machines has given designers, architects and artists tools for rapid prototyping. With the growth of Fab Labs and maker culture (Anderson 2012), larger communities are becoming empowered with the production means to make almost anything (Gershenfeld 2008). The Arduino open-source platform (Mellis et al. 2007) stands as a great example of a system that integrates a physical sensing/actuating device with a simple and intuitive development environment, designed to introduce novice makers to electronic prototyping. Personal fabrication projects (Baudisch and Mueller 2017) such as RoMA (Peng et al. 2018) demonstrate the potential of robot integration in ad hoc design-while-making systems.

Robotic fabrication in architecture has also seen a tremendous development in the last decade, with many architecture schools incorporating robotics as part of their curricula, and dedicated conferences promoting research in the field (Brell-Çokcan and Braumann 2013; McGee and Ponce de Leon 2014; Reinhardt et al. 2016; Willman et al. 2018). Projects such as Flowing Matter (Andreani et al. 2012) and the CEVISAMA pavilions (Seibold et al. 2018) explore the promising possibilities that design of robotic fabrication workflows may open up for innovative building structures. Additionally, art projects such as Mimic (Watson et al. 2017), Myro (2018) and Mimus (Gannon 2018) demonstrate the expressive potential that six-axis robotic arms have beyond their mere utilitarian implementations.

Even if industrial robotic arms are commonly used in these kinds of projects, they are still seldom implemented or controlled via the software tools that their vendors provide. The reasons for this are multiple. In addition

to the complexity and steep learning curves already discussed, these tools offer poor integration with common Computer Aided Design (CAD) and Manufacturing (CAM) environments typically used by these audiences. The recent popularity of visual programming languages for CAD such as Grasshopper3D (Rutten 2009) led to a bloom of robot-oriented add-ons, including HAL (Schwartz 2013), KukaPRC (Braumann and Brell-Çokcan 2011), TacoABB (Vogler 2016) or Scorpion (Elashry and Glynn 2014). These tools allow compilation of robot programs in native languages directly from the CAD environment, offering smoother learning curves and providing easier CAD to CAM workflows. However, they are in many cases licensed, vendor-specific and closed-source, making it hard to extend or customize for novice users.

Furthermore, these tools perpetuate the same old interaction paradigm that conventional engineering platforms provide: offline programming. They may allow visual programming, present a different syntax or feature better UIs, but ultimately, all decision-making is still designed at the digital stage, preceding offline execution of the post-compiled instructions. This makes it really hard to move beyond pre-canned instructions into concurrent control, and authors willing to engage with robots in an interactive way must develop their own ways into the machine, usually through hacks and custom tools. An example would be the ofxRobotArm package (Gannon and Moore 2016), an extension of the popular openFrameworks toolkit (Lieberman et al. 2018), which allows real-time control of UR5 arms in C++.

As a result of this circumstance, creative inquiry in the field of robotics is still in its very infancy. The problem is simple: the tools available to designers expect them to use the robot in the same way it has been used for the past half century. As a consequence, innovation and creativity are constrained to a fairly small domain, and thoroughly explored one. Breaking out of this model requires becoming a power-user, understanding the medium and gaining enough expertise to write bespoke software and, as it has already been discussed, this is not always a possibility —and should probably not have to be one anyway.

1.5 A Framework for Concurrent Robotics

In this dissertation, I propose a conceptual model called *Enactive Robotics* for the design of concurrent robot control systems. The main goal of this model is to blur the line between the system used to instruct the robot what to do and the one regulating such execution, integrating them into a single framework where *robot programs can be enacted concurrently during the design phase*.

This model proposes *actions* as the basic cognitive unit of interaction between a decision-making agent and a machine. An action represents a high-level request to change some of the properties of a mechanical device. The main characteristics of actions are that they are *platform-agnostic* —they are independent from concrete specifications—, *universal* —the same actions can be applied to different and multiple machines—, and *contextual* —their effect depends on the characteristics and current state of the device once applied. Additionally, Enactive Robotics crucially accounts for the large time scale-shift between concurrent

mechanical and computational runtimes[1]. Drawing inspiration from asynchronous programming, the model also introduces *events* as concurrent notifications to the decision-making agent about completion of significant stages in program execution.

The Enactive Robotics paradigm is built around the centrality of the *state* model as a *concurrent representation of the changing properties* of the devices participating in a system. The state model conforms the digital instantiation of the robotic environment with which a decision maker can interact via actions, and to which any physical device is a surrogate of. The central position of the state representation within the enactive model serves multiple functions, with the most important one probably being to serve as the *main interface between decision-making agents and mechanical devices*.

Actions and *events* conform the bidirectional information units of a concurrent, asynchronous, closed communication loop. Robot programs can be designed universally, and real-time control can be performed through the sequential issuing of action requests to a particular device. Program execution can be monitored based on the information received from the events, and on-the-fly decision-making is possible concurrently to program execution. This paradigm seeks to supersede the traditional, unidirectional offline programming model, and favor programming styles that are reactive to the changes in dynamic execution of the robot, rather than prescriptive about it. While this dissertation focuses primarily on industrial robotic arms, it will be proven that this model is suitable and can be applied to any programmable machine that performs motion in three-dimensional space.

The main thesis in this dissertation is that the proposed model towards *concurrent robotics* has two important advantages over traditional robot control systems. On the one hand, it provides an easier and more immediate entry point to robotics for novel users, as it provides an enactive, rather than symbolic representation of the machine, which aids the cognitive processes that lead to better understanding motion planning and control. On the other, it provides a framework with greater depth of possibilities for more advanced users, in which its real-time nature and immediate feedback facilitates experimentation, flow of thought and creative inquiry. This is also manifested by the fact that the offline programming model is a strict subset of possibilities within Enactive Robotics.

1.6 Structure

This dissertation begins by reviewing the notion of making things with digital tools in chapter 2, and recent work done in facilitating this process to wider audiences. Chapter 0 introduces the *Enactive Robotics* model for concurrent robotics, describing its general principles, conceptual architecture and applicability to machine control. To exemplify these principles, chapter 4 describes *Robot Ex Machina*, an open-source project that

[1] When describing a system in this text, the terms "run time," "run-time" and "runtime" will be used respectively to refer to the *time* needed for execution, a process that happens *during* execution, and the *platform* where execution takes place.

features two sample technical implementations of the model. The first one is "Machina.NET", a C# library designed for power-users and tech-savvy individuals familiar with building Windows applications, which demonstrates the full potential of the model contained within a single, unitary framework. The second one is an ecosystem of UI-based applications and utility libraries that connect robots to common design and development environments, geared towards novice and entry-level users. Chapter o features a collection of projects built by a community of advanced Machina users. The work is presented as case studies that demonstrate the capacity of the model to systematically enable deeper, richer and more complex interaction paradigms in robotic systems, inaccessible through traditional offline programming. Additionally, chapter o presents a controlled user test, where a comparative study was conducted on a group of novice robotic users. Participants were observed and surveyed during sessions of training and implementation of robotic tasks, using two different robot programming environments: a state of the art, "easy and user-friendly" interface provided by a popular robot vendor, and a system built after the principles of the Enactive model. The results of the study show evidence that the model is effective in providing a more intuitive entry point to robot programming, facilitating the cognitive processes that aid in its understanding. This dissertation concludes on chapter o with a summary of the contributions of the Enactive Robotics model, remaining challenges to address and future work.

1.7 Note on Style

This dissertation uses the following stylistic guidelines:

- Impersonal passive form is used to present the main scholarly elaboration of this work.
- First-person singular form is used to reflect the author's opinions and personal remarks.
- Third-person plural form is used as third-person singular gender-neutral form.

2 Context

In this section, a critical overview is presented of the context in which the work of this dissertation is situated. The perspective hereby offered draws from historical developments in the fields of computer science, programming, psychology and cognitive sciences, and aims to establish connections with their implications in contemporary state of the art technology-oriented practices in creative fields such as design, fabrication, architecture and art.

The chapter starts discussing the historical tendency of technology to foster role divisions in creative processes against the traditional role of craft, and how this bias has particularly affected machine control mechanisms. The role of digital interfaces as mediators between human-machine interaction is analyzed, and an overview will be presented of the lineage of cognitive theories that influenced their design. Subsequently, this account will zoom in the special case of programming systems for industrial robots, with particular interest in those tools with increasing popularity among the creative community. The range of interaction models that they exhibit will be evaluated, and a critical assessment on the need to systematically supersede them will be offered[2].

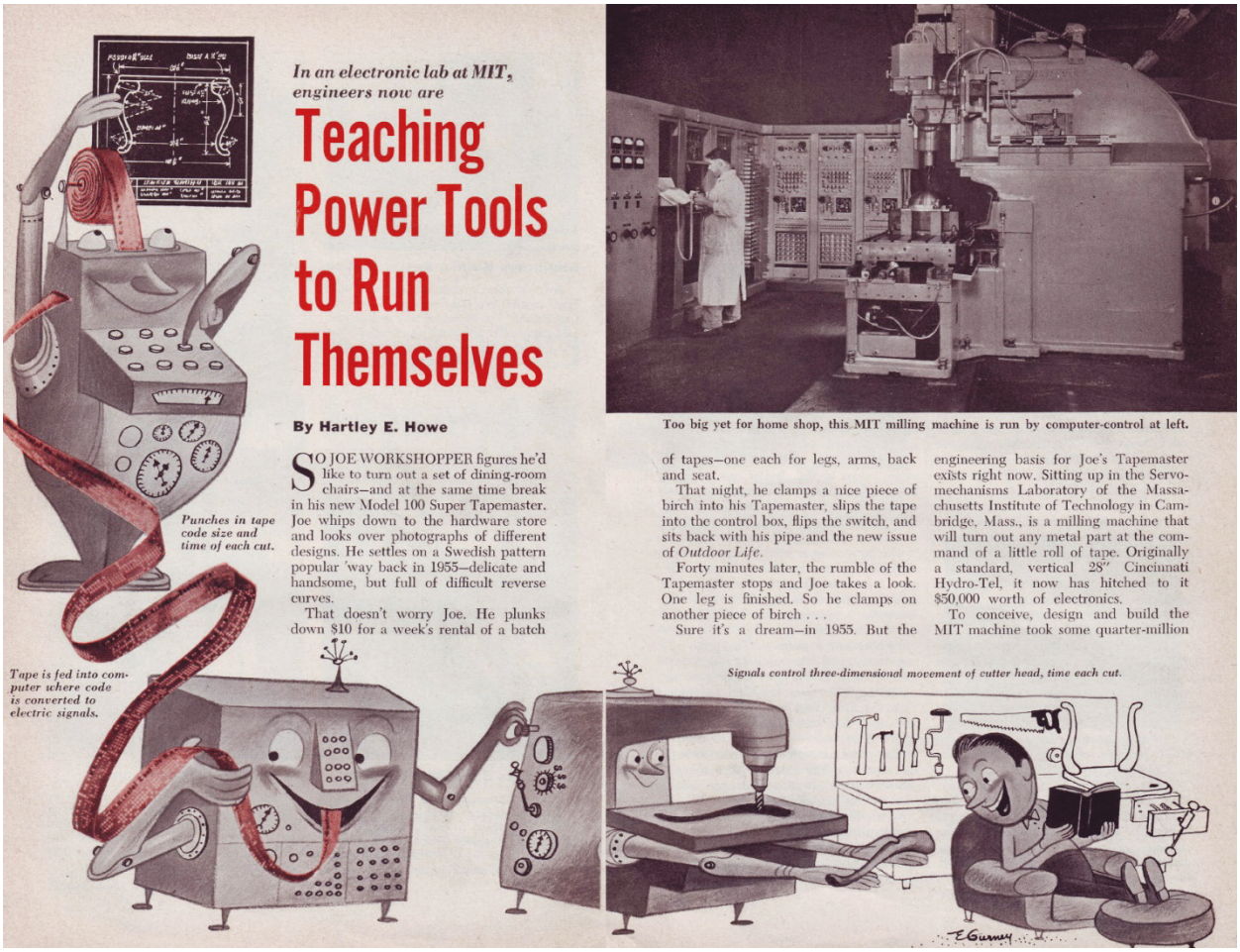
This historical account will help identify current challenges in robot programming and human-robot interaction, and serve as the starting point to situate the conceptual model proposed in chapter 3 for the design of concurrent robot control systems.

2.1 Designing and Making

The origin of programmable machines can be found on early attempts to customize the behavior of otherwise repetitive fabrication machines. The Jacquard loom is commonly regarded as the first of its kind, allowing pattern programming through the rather tedious and abstract process of card punching (Fernaeus et al. 2012). Advances in computation at Bell Labs and MIT and their application to fabrication technologies, especially

[2] An extensive account of several of the topics and sources discussed in this chapter can be found in the author's *Interfaces for Digital Making: Reflections at the Intersection of Creativity, Matter, Computers and Minds* (2016).

during World War II, would result in the first Computer Numerical Control (CNC) machines (Noble 2011). At the time, these machines were designed for their capacity to precisely and reliably fabricate non-mass-produced parts, and skepticism surrounded the idea of these "industrial giants [scaling] down to home-shop size and price" (Howe 1955). However, just like mainframe computers, the process of miniaturization, cheapening and democratization of fabrication machines has started a revolution which will eventually make them as pervasive as personal computers, with deep educational, economic and social implications (Gershenfeld 2008). Superseding mass production (Toffler 1981), generalized access to personal fabrication tools and the growth of the "maker movement" (Anderson 2012) are leading to what many are referring to as a "third industrial revolution" ("A Third Industrial Revolution" 2012).



In an electronic lab at MIT, engineers now are

Teaching Power Tools to Run Themselves

By Hartley E. Howe

SO JOE WORKSHOPPER figures he'd like to turn out a set of dining-room chairs—and at the same time break in his new Model 100 Super Tapemaster. Joe whips down to the hardware store and looks over photographs of different designs. He settles on a Swedish pattern popular way back in 1955—delicate and handsome, but full of difficult reverse curves.

That doesn't worry Joe. He plunks down $10 for a week's rental of a batch of tapes—one each for legs, arms, back and seat.

That night, he clamps a nice piece of birch into his Tapemaster, slips the tape into the control box, flips the switch, and sits back with his pipe and the new issue of *Outdoor Life*.

Forty minutes later, the rumble of the Tapemaster stops and Joe takes a look. One leg is finished. So he clamps on another piece of birch . . .

Sure it's a dream—in 1955. But the engineering basis for Joe's Tapemaster exists right now. Sitting up in the Servomechanisms Laboratory of the Massachusetts Institute of Technology in Cambridge, Mass., is a milling machine that will turn out any metal part at the command of a little roll of tape. Originally a standard, vertical 28" Cincinnati Hydro-Tel, it now has hitched to it $50,000 worth of electronics.

To conceive, design and build the MIT machine took some quarter-million

Too big yet for home shop, this MIT milling machine is run by computer-control at left.

Punches in tape code size and time of each cut.

Tape is fed into computer where code is converted to electric signals.

Signals control three-dimensional movement of cutter head, time each cut.

E. Gurney

Figure 2 - "Too big yet for home shop, this MIT milling machine is run by computer-control at left." Image from Howe (1955).

Theoretically, everyone nowadays should be able "to make (almost) anything." However, it does not seem to be exactly the case. And while the reasons are probably to be found on a complex network of interrelated socio-economical factors, the key one that this dissertation aims to address is the barrier that software interfaces usually represent for users willing to program these machines. Howe (1955) would early on point out that, despite the fact that CNC machines would make it possible for anybody to fabricate their designs at home, the

production of the "blueprints" would remain a specialized technique not likely to hit the masses. Stan Davis, widely regarded as having coined the term "mass customization" in his 1987 *Future Perfect*, would acknowledge that, while technologies were already in place at the time, "the bottleneck will once again be software" (Davis 1996, p. 156).

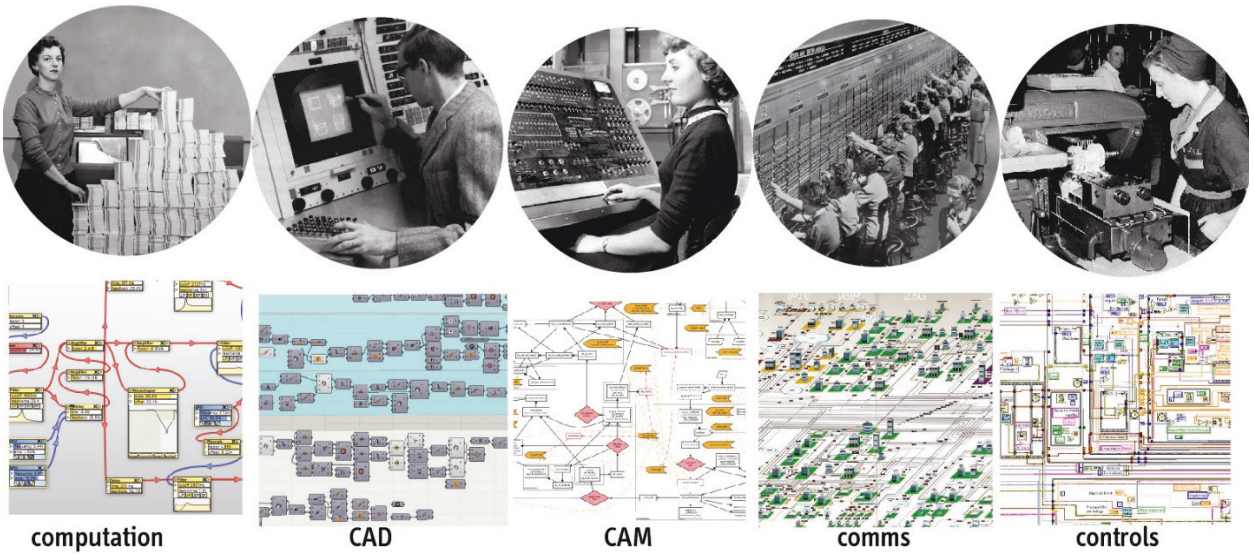


Figure 3 - "The historical separation of roles into computer, drafter, toolpath planner, machine interface, and machine control no longer apply." Image from Peek (2016).

In this context, it is important to discuss the role that digital interfaces play in relation to design and physical making processes. Traditional definitions of *craft* carried a strong sense of material making, loaded with connotations of special skill and knowledge. Craftsmanship stood for the holistic process of creation—from its initial conception to the final material realization. So unitary and concentrated was this process that the notion of *design* didn't exist detached from that of *making*, both embodied in the hands and the mind of the craftsman, and elevated to the art of *craft* (Sennett 2008). Or at least, such was the case until the Renaissance. Scholars trace the beginning of the split between *designing* and *making* to the works of Leon Battista Alberti (Ingold 2009; Carpo 2011). In his 1458 *De Re Aedificatoria* (1988), he advocated for the necessity to create a formal notational system for architectural representation which, as opposed to traditional perspectival views, could be unequivocally measured to scale and used as identity representations of the building's materiality. The purpose of Alberti's *lineamenta* was to free the architect from the material aspect of building, delegating this task to onsite masters commanded by this comprehensive set of instructions, with no room for interpretation or improvisation. Such a division between the "mental projection of form" —design— and its material development would start gradually permeating all forms of craft, from architecture to product manufacturing and printing, accelerated by advances in descriptive geometry and mechanical reproduction (Carpo 2011).

Modern advances in material science engineering (C. Smith 1968), and in particular, the creation of homogeneous materials, would only widen this breach. Modern CAD packages provide interfaces for the

creation of abstract, ideal and material-less geometry, working under the assumption that it will eventually be materialized into some form of "obedient matter" (De Landa 2002, p. 134). This model fosters an authoritarian, form-imposition philosophy that negates the morphogenetic agency of materials (Deleuze and Guattari 1987), generates "further disengagement of ideas and matter" (Mitchell and McCullogh 1995, p. 464) and detaches the creator from the "textility of making" (Ingold 2009). Mastery of digital design tools became a professional skill —just like control of digital manufacturing tools is increasingly becoming a highly specialized domain— with development in each field growing increasingly solipsistic rather than trying to building bridges between each other. The historical inertia of machine design still perpetuates obsolete role divisions between program creator and user, only accentuating the problem (Peek 2016). It is probably no coincidence, for example, that scholars argue that the next frontier of research in the field of design robotics is software (Bechthold 2010; Braumann and Brell-Cokcan 2011).

A split approach to making has resulted in specialized professions tackling the issues of each party involved separately, leaving the layman in a difficult situation. Far away from the times when the craftsman employed his mental and manual skills into a holistic creative process, the digital maker is challenged by the great investment, difficulty and commitment necessary to access contemporary digital design and manufacturing tools. And too often deterred by them.

> Training most of the public in the skills of industrial design is simply not viable. [...] Design tools that can be used by everyday users to create customized products is the solution to the specification problem in personalized fabrication. In fact, it is probably the most important technical problem that must be solved before the vision of personal fabrication can come into being. (Landay 2009, p. 80)

If an effort is to be made to truly democratize access to digital design and fabrication tools, the emphasis should lay on the software counterpart of these elements. The modern split between designing and making, as well as the role division that programmable machines typically foster, will serve in this dissertation as the basis to frame the forthcoming discussion around the dialectic between *programming* and *controlling* machines and, in particular, the common interaction paradigms that software tools provide to perform them.

2.2 Cognition

The theory, design and implementation of digital interfaces has traditionally been greatly influenced by developments in psychology, epistemology and cognitive science. During the last decades of the 20th century, a wealth of research focused on unfolding the central role of computers as mediators between humans and machines. Moreover, and due to the computer's neutral character as an all-purpose machine (Nelson 1975), it will be argued in this dissertation that it is the *software* running on those computers —the intangible collection of computational procedures that regulate the streams of information running through them and how users interact with it— which will constitute the main *interface* between the agents involved in human-machine interaction, most prominently defining its nature.

The interface constitutes the medium where different entities meet, with special focus on the interaction between them. The way the interface will mediate the relation between the participants will strongly depend on their nature: "The shape of the interface reflects the physical qualities of the parties to the interaction (the interactors, if you will)" (Laurel 1990, p. xii). Particular attention was drawn to the capacity of the interface to convey certain opportunities through "visual affordances" (Norman 2013), a term adapted from Gibson's ecological psychology (1977). Similarly, the importance of the user as the main subject of inquiry was manifested by the formalization of techniques to characterize them (Gomoll 1990; Rheingold 1990; Hoare 1973), and even on the manifested advocacy against their de-personalizing through the very same term "user" (Lialina 2012).

In an effort to address to users' needs and provide easier, more natural interactions with computers, interface designers required tools catered to humans' cognitive capacities and, during the early days of human-computer research, *metaphors* became the dominant form of intellectual aid. Metaphors were the designer's proposal of a conceptual model for appropriation of the interface's interaction principles: they provided a reference framework of behaviors and expectations, which the user could relate to by analogy with a familiar image evoked in the metaphor. Metaphors constituted powerful tools to use familiar situations in order to help navigate novel ones, such as through the use of "windows," "folders" or the "desktop" (Mountford 1990), or even establishing "icons" as "visual metaphors for concepts" (D. Smith 1975). Metaphors became so widely and overwhelmingly adopted as the "holy grail" in interface design (Erickson 1990), that certain authors started raising awareness about the potential problems of this model. The main argument against it was grounded on the difficulties caused by the differences between the user's mental model and the allegory (Erickson 1990; Mountford 1990). But moreover, scholars would raise awareness about the constraining effects that imposed metaphors could have in creative solutions to interface design, like in situations where "slavish adherence to a metaphor prevents the emergence of *things that are genuinely new*" (Nelson 1990, p. 239). Instead, it could be exactly in the "magical" properties of the interface, those that differ from the real thing, where the potential for creative solutions could be found, and where the most attention should be paid to (Kay 1990).

Metaphors became so popular because they constituted a pre-designed mental model, a helpful abstraction for the user to understand an otherwise unfamiliar system. Designing good metaphors was fundamentally an effort from interface designers to aid the cognitive principles involved in human knowledge acquisition. The design of these structures was deeply rooted in the understanding of the thought processes that were involved in the ways human minds perceive their environment, and the alleged techniques used in comprehending it. In this sense, most interface design scholars trace back the foundations of their cognitive approaches to Swiss psychologist Jean Piaget (Gruber and Vonèche 1977), commonly attributed with establishing the *constructivist* theory of cognitive development. Piaget argued that humans produce knowledge and form meaning in relation to the interaction between their experiences and their ideas. The fundamental contribution in Piaget's constructivism is the notion that the way the individual understands reality is through the *construction of mental models* —such as, for instance, a metaphor. These models then serve to explain what

he perceives through his experience, and to notice the discrepancies between what he already knows and what he discovers in his environment:

> Piaget has sometimes labeled his position constructivism, to capture the sense in which the child must make and remake the basic concepts and logical-forms that constitute his intelligence. Piaget prefers to say that the child is inventing rather than discovering his ideas. This distinction separates him both from empiricism and from apriorism. The ideas in question do not preexist out there in the world, only awaiting their discovery by the child: each child must invent them for himself. (Gruber and Vonèche 1977, p. xxxvi-)

Figure 4 - "I begin by describing one example of a constructed computational "object-to-think-with." This is the "Turtle"." Image from Papert (1980).

Piaget's pioneering work would have a tremendous impact in all fields of cognitive science —with particular importance in education— and would influence generations of scholars. One of these preeminent figures would be mathematician, computer scientist and educator Seymour Papert. Papert would subscribe the postulates of constructivism, but would place special emphasis on the educational implications of this model. He would extend the postulates from Piaget —his mentor— into his own theory, *constructionism*, whose main contribution was the individual's conscious engagement in the activity of constructing something as a way of learning: "[...] constructionism boils down to demanding that everything be understood by being constructed" (Papert and Harel 1991, p. 2). Many authors have summarized the principles of constructionism in the "catchy" phrase "learning-by-making," although Papert himself would deem the term as reductionist: "[I suggest to elaborate] a sense of constructionism much richer and more multifaceted, and very much deeper in its implications, than could be conveyed by any such formula" (p. 1). Papert would go on to develop an impressive body of work in the field of computer-enhanced education, with particular relevance of his LOGO programming language and its application to turtle robotics (Papert 1980).

Another prominent figure in the field of cognitive studies would be American psychologist Jerome Bruner. In his research, Bruner built much of his developmental theories drawing from Piaget's constructivist philosophy, deeply involved as well in the study of children's learning processes and the way they develop as they grow.

While he subscribed the epistemological model based on experience and the construction of mental models, Bruner went a step further and proposed a three-stage process to understand the modes in which these models are represented. From his point of view, humans construct these models first by *enactive* representation, that which is acquired by direct manipulation and interaction, as in a kid playing with a toy car. On a second stage, knowledge is created by *iconic* representation, that which is founded in visual and perceptual principles, as in images or videos of cars. And finally, the ultimate form of learning is acquired by *symbolic* representation, that which is governed by abstractions, such as words or language ("Theory of Instruction" 1966, pp. 9-12). Bruner discusses how a full epistemological cycle to the unknown starts with direct interaction and experimentation with the subject matter, evolves into the capacity of completion, extrapolation and prediction based on perceptual organization, and is fulfilled when such knowledge can be transmitted and expanded through symbols. Such a staged approach to the representational nature of learning would later inspire Kay to coin his own slogan "Doing with Images makes Symbols," to imply that "one should start with—be grounded in—the concrete “Doing with Images,” and be carried into the more abstract “makes Symbols”" ("User Interface" 1990, p. 196).

However, while Bruner agreed with the constructivist postulates of self-motivated education and discovery, he believed that this model also allowed room for an instructional approach to education, where the learner could be capable of learning any content as long as it was adequately structured to be apprehended according to the three modes of representation. He would suggest that a completely unstructured approach to learning based on exploration —such as the one suggested by Papert— may not suit all intellectual styles equally. "There is a tendency to assume that all we need to do is provide a good tool and the child's natural curiosity and motivation will take over. [According to Brunner] “Discovery favors the well-prepared mind”" (Nicol 1990, p. 114, quoting Bruner 1962).

Bruner's epistemological theory is of especial interest in the context of this dissertation, and in particular, his proposal of direct interaction with the environment as the earliest mode of knowledge acquisition in children development. Recent currents of epistemological thought, influenced by Bruner and the phenomenological dimension of his model (Merleau-Ponty 1962), have elevated *enaction* as a new paradigm in the cognitive sciences. The term was coined by Varela et al. (1991) to describe a cognitive process that is driven by *embodied action*:

> We can now give a preliminary formulation of what we mean by enaction. In a nutshell, the enactive approach consists of two points: (1) perception consists in perceptually guided action and (2) cognitive structures emerge from the recurrent sensorimotor patterns that enable action to be perceptually guided. (p. 173)

Their notion of *enactivism* is deeply rooted in the biological and cultural contexts of living organisms, who construct conscious meaning by actuating their environment. The idea that cognition is enacted —brought forth through action— draws from constructivist theories of knowledge representation, but places a strong emphasis on the individual's capacity to proactively interact with their medium: "cognition is the exercise of skillful know-how in situated and embodied action" (Thompson 2007, p. 13). After all, "intelligence cannot

merely exist in the form of an abstract algorithm but requires a physical instantiation, a body" (Pfeifer and Scheier 1999, p. 649).

Enaction as a new paradigm for cognitive science (Stewart et al. 2010) has recently drawn increasing attention from fields interested in its applicability as a new model for embodied autonomous systems, such as developmental robotics (Cangelosi and Schlesinger 2015) or artificial intelligence (Froese and Ziemke 2009). However, in the context of this dissertation, enactivism will be examined as the foundational cognitive tool towards the understanding and enaction of machine control, suitable to capitalize on a robot's affordance as a tangible medium to interact with through guided action.

2.3 Robot Programming

The design of digital interfaces is at the core of research in human-computer interaction, and constitutes an effort in unfolding the nature of the relationship between these two significantly different agents. While arguably more complex than CNC devices, computers are fundamentally programmable machines as well, and the development of the techniques to instruct them may serve as the foundation to build new models of robot programming and control.

Inspired by Jacquard's work, the origins of the computer may be traced back to Babbage's analytical engine, and Lovelace's notion of the *programmability* of the machine's function through rules of computation (Gleick 2012). While the engine was never completed, if became the conceptual model that led to the development of the first electro-mechanical computers after World War I. However, these early machines still relied heavily on mechanical processes to perform their function, and their programmability was conditioned to the physical rewiring of their control systems. It was Von Neumann who would first propose a system architecture where the computer program would be stored in the computer's memory along with the data ("Report on the EDVAC" 1945), as opposed to the program residing on an external system. The advantages of the Von Neumann architecture turned it into the de facto model for modern computers, and could perhaps be considered the origin of the modern divide between computer *hardware* and *software*.

The conceptual —and early on, even physical— split between a machine's behavior and its control mechanisms meant that designing the function a computer finally superseded the mechanical engineering domain, becoming a conceptual endeavor detached from the physical dimension of the machine. Or at least, that was the theory. Early computer programming was still bound to the composition of tedious processor-specific instructions called "machine code" (Petzold 2000), that required significant domain knowledge and expertise. An immediate response from the scientific community was born, advocating for the design of systems "to make computers more conveniently usable, to make them less of an exclusive domain of the highly trained specialist" (Wirth 1974, p. 23), hence triggering a wealth of research and experimentation in the novel field of computer programming. Prominent figures in this conversation would be Turing Award laureates Niklaus Wirth, Edsger

Dijkstra and C.A.R. Hoare, with seminal writings in which they argued for the capacity of programming languages to assist the programmer in the design process (Hoare 1973), advocated for the need of higher abstraction levels in order to do this (Wirth 1974), and even compared the relationship between a programmer and their tool to the notion of craft:

> The tool should be charming, it should be elegant, it should be worthy of our love. This is no joke, I am terribly serious about this. In this respect the programmer does not differ from any other craftsman: unless he loves his tools it is highly improbable that he will ever create something of superior quality. At the same time these considerations tell us the greatest virtues a program can show: Elegance and Beauty. (Dijkstra 1962, p. 10)

After more than fifty years of development in the field of computer science, far seem the days of "machine code." Modern programming languages such as C/C++/C#, Java, Python or JavaScript constitute platform-agnostic frameworks that provide high-level abstractions for computer programming, with machine specificity taken care of by post-compilation processes that usually remain hidden to the average programmer. The expectation nowadays is that a program written in any of these languages will successfully execute on any kind of computer, and consistently yield the same outcome.

Of particular interest to this work is the special case of programming languages designed specifically for creative communities. Early work on the expressive potential of computer graphics in design and art (Maeda 2001) led to the creation of Processing (Reas and Fry 2007) and its sibling project p5.js (McCarthy 2018), both incredibly popular frameworks among visual artists, with a strong emphasis on education and community building (Processing Foundation 2018). In a very similar spirit, although arguably for more proficient users, openFrameworks (Lieberman 2018) provides a powerful environment for creative coders, with particular emphasis on 3D graphics and interactivity. And moreover, the popularity of the Arduino open-source platform (Mellis et al. 2007) has provided an accessible entry point to amateur electronics to a large community of tinkerers and creative individuals. While none of them is technically a full programming language on its own[3], they provide such an opinionated and consistent interface for the features they enable, that for most practical purposes, they are often framed as such.

Perhaps it would be fair to assume that advances in computer programming would have had an impact on the development of improved systems for machine programming. Unfortunately, this has not been the case. Contemporary machine design is largely the same as 50 years ago, despite progress in other fields such as computer science and network engineering (Peek 2016). For instance, most CNC machines nowadays are still controlled by the same old programming language known as G-Code. This protocol is only able to support "one-way information flow from design to manufacturing" (Xu and Newman 2006), and even though it is regulated by an international standard (ISO 6983-1:2009), the interpretation is so open that most modern CNC machines use particular flavors of this language, hampering interoperability and cross-platform compatibility.

[3] These frameworks are in fact libraries for the Java (Processing), JavaScript (P5.js), C++ (openFrameworks) and C (Arduino) programming languages respectively.

Finally, the creation of G-Code programs depends on the capacities and compatibilities between CAD/CAM workflows, and as such, usually limited by their integration:

> Vendors and users are still seeking a common language for CAD, CAPP, CAM, and CNC, which integrates and translates the knowledge of each stage with no information loss. Though there are many CAM tools supporting NC manufacture, the problem of adaptability and interoperability from system to system was and is still seen as one of the key issues in limiting the wider use of these tools. (Xu and Newman 2006, p. 142)

The problem only worsens when considering the case of industrial robots. As discussed in §1.1, robot manufacturers usually provide proprietary frameworks for robot programming and control. These environments typically require significant training and domain expertise, and are oriented for highly specialized audiences in the engineering and manufacturing fields. Furthermore, each vendor requires the use of a different language for programming their robots, accentuating the problem of adaptability and interoperability. Efforts have been made to open up the field of robotics to the research community (Quigley et al. 2009; Pan et al. 2012), but the target audience remains a reduced group of highly skilled individuals tied to engineering fields. Without the proper training and time investment, robotics is still out of reach to more general audiences (Biggs and MacDonald 2003).

Fortunately, some progress has been done in the last few years to bring creative communities closer to the world of industrial robots, with the special case of the "RobArch: Robotic Fabrication in Architecture, Art, and Design" conference series (Brell-Çokcan and Braumann 2013). The work presented in these conferences shows the tremendous popularity of the tandem between the Rhinoceros NURBS modeling environment 9McNeel 1993) and the Grasshopper visual programming environment (Rutten 2009) to constitute the main CAD platform for robotic fabrication workflows, with an estimated 81% of the projects featuring them at their core[4]. This popularity has led to the development of multiple robot programming tools that integrate in this environment. Among these, HAL (Schwartz 2013) and KukaPRC (Braumann and Brell-Çokcan 2011) are arguably the most popular, with roughly 25% of the work presented in RobArch built around these tools. HAL and KukaPRC, provide a collection of visual components to describe typical entities in a robotic program, such as variables, targets, configuration, etc., as well as basic simulation tools to preview machine behavior. Other tools provide similar functionality, such as SuperMatterTools, Mussel (Johns 2014), TacoABB (Vogler 2016) and Scorpion (Elashry and Ruairi 2014). It could be argued that the proliferation of industrial robotic arms in fabrication facilities at universities around the world (Gramazio and Kohler 2014), as well as the availability of such tools to bridge the gap between CAD modeling and CAM implementation, have initiated a trend toward exploration in robotic fabrication outside the engineering fields.

[4] The data presented in this section regarding the use of software RobArch conferences is based on a survey by the author based on the work published in Brell-Çokcan and Braumann (2013), McGee and Ponce de Leon (2014), Reinhardt et al. (2016) and Willman et al. (2018). The survey includes all the papers published in the science, projects and workshop sections, and quantities are based on the percentage of these papers where the reference metric could be determined or reasonably estimated.

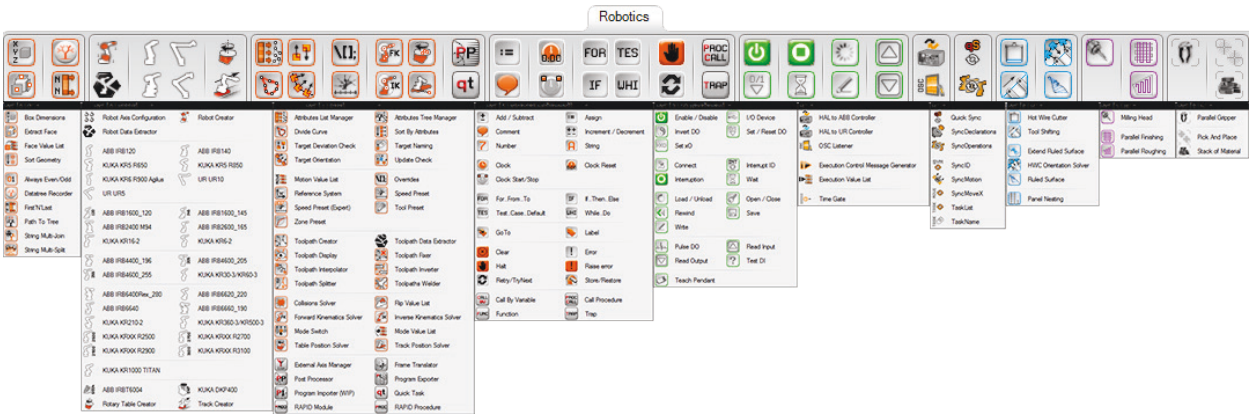


Figure 5 - API of Grasshopper components in HAL (Schwartz 2013): the language and logics are closer to their internal machine representations, rather than to the high-level actions they serve.

However, challenges still remain within this new wave of robot programming for designers. Many of these tools provide a model of features and options whose language and logics are closer to their internal machine representations, rather than to the high-level actions they serve (Figure 5). In this sense, the problem of cognitive accessibility is seldom addressed, as their robot-code-like interfaces demand symmetric robot-domain-knowledge from the user in order to properly work with them; in many instances, users could be better off just learning how to write native robot code by themselves. Such domain knowledge is often vendor-specific as well, as some of these tools only work for ABB robots —Mussel, TacoABB—, Kuka robots —KukaPRC— or Universal Robots —Scorpion—, requiring the rewriting of workflows with different tools to adapt them between systems. Furthermore, most of these tools are closed-source —all except Mussel and Scorpion—, and some of them requiring costly licenses to use them —HAL, KukaPRC—, becoming an obstacle for creative exploration, extension and customization. But moreover, while some of these frameworks implement basic real-time functionality to automate program streaming to the robot —HAL, KukaPRC, Scorpion—, their design is fundamentally geared towards the creation of offline programs. As it may be discussed in the following section, this bias perpetuates an obsolete model of human-machine communication that fundamentally constrains the creative capacities inherent in human-robot interaction.

This scenario of state-of-the-art technologies, platforms and tools for robot programming and control among creative users will be used in this work to identity the target community that this dissertation is trying to address, to justify the choice of platforms selected for its conceptual and technical contributions, and to establish a benchmark to be assessed against.

2.4 Concurrency

It might be useful at this point to take a small step back, and reflect on what the notion of a *robot* is. This question is usually the subject of endless philosophical debate, ranging from simple definitions such as "a machine capable of automatically carrying out a complex series of movements, especially one which is

programmable,"[5] to increasingly ontological queries on their nature: "a robot is a constructed system that displays both physical and mental agency, but is not alive in the biological sense" (Richards and Smart 2013).

In any case, most authors agree that a robot is essentially constituted of two parts: a physical manifestation that actuates and/or senses its environment, and a logic system that drives its behavior. And while the former, hardware design, portraits a whole set of challenges on its own, the work described in this dissertation tries to address the challenges present in the latter, the creation of the software mechanisms to program and control robotic devices.

A more practical definition of the nature of robots is offered by Matarić (2007):

> A *robot* is an autonomous system which exists in the physical world, can sense its environment, and can act on it to achieve some goals. (p. 2)

In the context of industrial robotic arms, it is clearly noticeable that the "robotic" character of these machines falls rather short under this framing. Their standard application is as physical actuators, but their autonomy is constrained by predefined tasks that seldom implement any kind of sensing or external inputs, hence lacking a sense of overall goal through autonomous decision making.

The main hypothesis presented in this work is that the problem does not lay on the machines themselves, but rather on the paradigms used to control them. Industrial robots, like most other CNC machines, are commonly programmed under the offline model, where digital simulation tools external to the device are used to predesign its behavior, encode it into a file in the machine's native language, and execute it offline, independently from the system that created the file (see §3.2). While this system has clear advantages for automation systems with large product volumes, "the complexity of programming remains one of the major hurdles preventing automation using industrial robots for small and median enterprises" (Pan et al. 2012). In this dissertation, the scope of this hurdle is extended to include prospective machine users in the creative fields, facing the tremendous entry barrier that these programming systems pose for them.

[5] Oxford English Dictionary, Third Edition, June 2000.

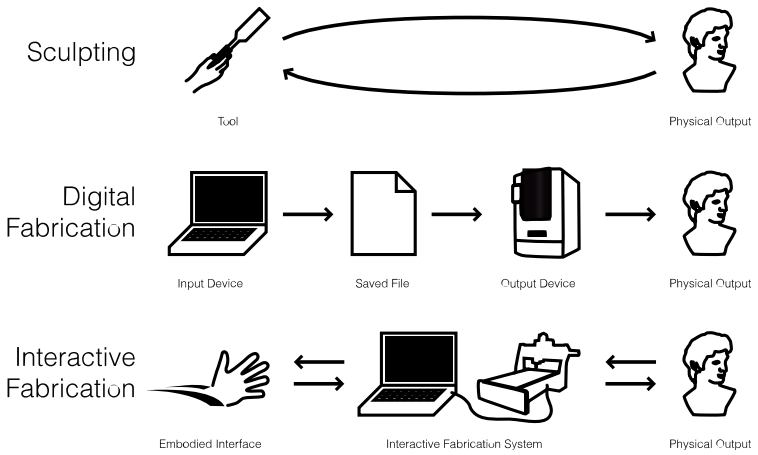


Figure 6 - "Interactive Fabrication contrasts with existing approaches by allowing real-time input to digital fabrication." Image from Willis et al. (2011).

Several authors have also identified the constraining interaction model that offline programming constitutes for creative exploration in digital making. Willis et al. (2011) propose "Interactive Fabrication" in their eponymous paper as a new paradigm in human-machine interaction, where the human is situated back at the center of fabrication processes, and has the capacity to concurrently interact and guide the machine during its work. The model aims to supersede the unidirectionality and detachment of current fabrication methods, bringing the crafter back in direct touch with the fabricated output (McCullough 1996). Recent work on "personal fabrication" (Mueller 2016) showcases the capacity of this model to provide richer and deeper interaction experiences with programmable machines, with projects such as "Constructable" (Mueller et al. 2012) or "RoMA" (Peng et al. 2018).

Recent developments in robot programming systems reflect these aspirations. Biggs and MacDonald (2003) suggest "instructive systems, [those] given a sequence of instructions in real-time" as the "highest level of programming" (p. 7), and identify that in software development for machine control, "the strongest trend is the addition of intelligence to programming systems" (p. 8). Matarić (2007) delineates the role of robot programming languages as dependent of the control architectures of the machine, and proposes four different paradigms: deliberate, reactive, hybrid or behavior-based control[6]. Finally, Pan et al. (2012) argue that, as the development of more powerful CAD systems, computer vision, sensors, etc. are fostering the development of new programming methods that incorporate them at their core, "the boundary between online and offline programming methods are becoming blurred" (p. 1). The advocacy for novel hybrid systems that combine deferred and real-time programming will be of particular importance in this context, as the model contributed in this dissertation is discussed in §3.3.

Returning to the RobArch conference series as a case study of representative use of robot programming tools, a clear tendency is observed. An estimated 62% of the work presented in these conferences was implemented

[6] It is noteworthy that she doesn't recognize offline control as an option because, as seen from her earlier definition, this would nullify the robotic nature of the machine.

with offline systems, with only 23% implemented closed control loop[7]. Relevant examples of the latter include projects accounting for tolerances in construction in near real time through "adaptive part variation" (Vasey et al. 2014), control of material feedback during production (Raspall et al. 2014), sensory-informed autonomous robotic assembly (Jeffers 2016)] or its connection with agent-based generative systems (Snooks and Jahn 2016). It is noteworthy how several of these authors strongly advocate for the implementation of closed feedback loops in robot control systems. Amtsberg et al. (2016) describe the crucial effect of implementing sensory information in their work, concluding that "the implementation of feedback enables new perspectives on project-relevant production techniques" (p. 313). Snooks and Jahn (2016) suggest "replacing empirical testing and calibration with feedback loops that self-correct and self-stabilize over time" (p. 228). Ultimately, Raspall et al. (2014) note:

> In a similar way as HAL, Rhino2krl, and Your expanded robotics to a sizeable number of designers, similar efforts to systematize and simplify programming of feedback control will surely open this approach to larger audiences. (p. 341)

However, efforts to systematize concurrent robot control and bidirectional data exchange have so far been discrete. While 41% of the offline systems published in RobArch was built using one of the tools described in §2.3, only 28% of the online systems used them as part of their architecture. 79% of the projects featuring closed control loops relied on the development of custom solutions, usually in the form of Python scripts and manually handled low-level machine communication (Dörfler et al. 2012; Keating et al. 2014; Bard et al. 2018)]. Unfortunately, the domain knowledge and skill necessary to develop these techniques prevented the authors of other projects from synchronizing their Arduino board synchronously with robot execution (Friedman et al. 2014), using live sound to drive the fabrication process (Reinhardt et al. 2014) or incorporating feedback beyond the user's visual perception (Yuan et al. 2016), resonating with the concerns expressed by Peek (2016).

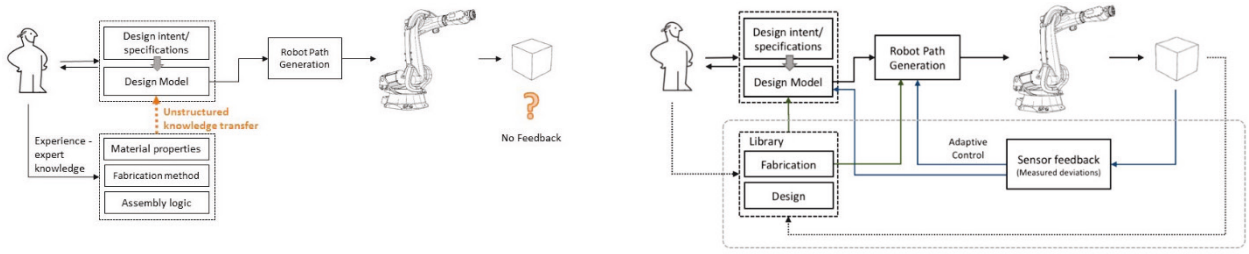


Figure 7 - "Current one-directional design to robotic fabrication workflow" (left) vs. "proposed bi-directional design-fabrication workflow" (right). Images from Sharif et al. (2016).

Attempts to systematically streamline this process have been conducted at many levels, with varying degrees of success. Xu and Newman (2006) advocate for the STEP-NC protocol of functional programming for CNC machines to supersede the obsolete G-Code, so that the "program can separate the "generic" manufacturing

[7] See note 4. The remaining 15% could not be estimated or do not describe a particular implementation. Interestingly, most of the accounts describing online robot developments come from workshop reports rather than scientific papers.

information (what-to-do), from the manufacturing information (how-to-do) that is native to a specific machine tool" (p. 146). However, more than a decade later, G-Code is still the leading protocol for much of the CNC world. Shani et al. (2016) propose "adaptive control" to systematically incorporate humans on the loop at the cornerstone of "a framework for a comprehensive (computer and human) feedback system to be integrated into the design to fabrication workflow" (p. 3). Unfortunately, this advocacy remains at the conceptual level, with no technical implementations illustrated as samples of the model. Shahmiri and Ficca describe a "model for real-time control of industrial robots" (2016), and propose a technical architecture and sample implementations. Nevertheless, their contribution is specific to the Grasshopper environment and ABB robots, and not generalizable to other platforms. Raspall (2015) presents a model for responsive design and fabrication in contexts with elevated uncertainty, including a comprehensive architecture of robotic real-time control mechanisms and extensive elaboration of low-level code to support it. Sadly, his contribution is specific to ABB robots and the particular fabrication case he addresses.

The work presented in this dissertation situates itself at the end of a decade-long conceptual discussion and advocacy for improved modes of machine control to supersede obsolete offline programming paradigms, and hopes to trigger a new wave of applicable contributions for more accessible, systematic and generalizable models to close back the loop, and foster smart, adaptable, responsive and collaborative robotic systems.

3 An Action-State Model for Enactive Robotics

Industrial robots were originally designed for manufacturing applications and, as a logical consequence, the software programming and control tools that ship with them cater to the needs of the community created around them: robotic engineers, developers, integrators and implementers. There is an abundance of tools that capitalize on the technical expertise of this group, and are optimized for the typical use-case scenario of these machines: predictable repetitive tasks on a factory's assembly line.

However, for anyone else outside of this field, two major barriers still stand in the way to increased adoption and creative experimentation. Learning how to use a robot can be a rather daunting enterprise, requiring extensive technical training in the details of how they are built and operate. But moreover, exploring novel applications for these machines demands an additional leap into the technology ladder, as it often requires developing custom regulating mechanisms that allow the user to control the robot in a different way from the one it was designed for. Unfortunately, significant mastery is needed of the nuances of the device at hand, and ad-hoc solutions become hardly generalizable across wider families of platforms. How can the immediacy of such a tangible medium contribute to lowering the cognitive entry barrier to program it? Do feasible alternatives to the offline control paradigm exist? Is it possible to conceive a more fluid interaction model with robots? This research tries to propose a systematic response to some of these questions.

This dissertation introduces *Enactive Robotics*, a conceptual model for the design of concurrent robot control systems. This research assesses the typical divide between robot programming and control environments as fostering a critical detachment between the creator and their tool, and advocates for an integrated model where programs become *enacted* on the robotic system, blurring the distinction between the thinking and doing stages. Such model is grounded on the capacity of a decision-making agent to interface with the machine in real time via a set of high-level requests named *actions*. Actions conform the atomic units of cognitive interaction with the robot, and their effect on a particular device is dependent on the device's nature and state. Furthermore, this paradigm crucially involves considering two aspects of the critical differences between the controller and the controlled: the large-scale shift between the computational run time of the control system and the mechanical execution time of the machine, and the high-level mental definition of an action versus the low-level implementation of its effects on the robot. The model proposes the central role of a

system *state* representation as mediator between both parties, maintaining concurrent descriptions of the dynamic properties of all the agents connected to the system, and acting as interpreter between their internal representations of the outcome of an action.

The main thesis in this dissertation is that robot control systems designed following the principles of the enactive robotics model feature two important advantages over traditional ones. On the one hand, they provide an easier and more immediate entry point to robotics for novel users, as they provide an enactive, rather than symbolic representation of the machine, which aids the cognitive processes that lead to better understanding motion planning and control. On the other hand, they provide a framework with greater depth of possibilities for more advanced users, in which its concurrent nature and immediate feedback facilitates experimentation, flow of thought and creative inquiry. This is also manifested by the fact that the offline programming paradigm as an interaction model is a strict subset within the enactive robotics model.

In this section, the enactive robotics model for the design of concurrent robot control systems is introduced. The chapter starts by defining the agents and systems involved in the model through comparison with traditional robot control environments, and describing the prerequisites necessary for a successful implementation. Borrowing from cognitive theories in enactive knowledge acquisition, actions are proposed as the basic unit of interaction with the system, and their fundamental properties are described. A central state representation of the system is then presented, and its role as mediator between the different runtimes and action representations is discussed.

The conceptual model laid out in this chapter will be the guiding principle for the sample technical implementations described in chapter 4, and evidence of the claimed advantages of this model will be presented through case studies and user testing in chapters 0 and 0.

3.1 Definitions

In the context of this dissertation, a *robot programming system* will be defined as a digital environment used to translate high-level input data structures into low-level machine instructions, intended for operating a *programmable mechanical device*. Such device is any machine able to perform tasks that actuate or sense their physical context, and whose behavior is customizable through *programs* in the form of command sets like, for example, computer code[8]. The programming system typically features interfaces and tools that allow a *programmer* to create or modify the input parameters, design the program sequence and customize the details of their machine interpretation. The translation from high-level inputs to native machine instructions is usually referred to as *compilation* or *post-processing*.

[8] This dissertation will focus mainly on six-axis industrial robotic arms as programmable mechanical devices. However, it will be discussed in §3 how the enactive robotics model is applicable to any machine that performs motion in three-dimensional space, like 3D printers, routers, laser cutters, drones, etc. This definition explicitly excludes devices that do not typically interact with their environment such as, for example, computers themselves. The reasons behind this differentiation will be discussed in §3.5.

Relevant examples of robot programming systems are discussed in chapter 2. In the case of six-axis industrial robotic arms, tools such as HAL Robotics (Schwartz 2013) or Scorpion (Elashry and Glynn 2014), in partnership with Rhinoceros (McNeel 1993) and Grasshopper3D (Rutten 2009), are used to transform CAD data into spatial toolpaths, actuation procedures and, ultimately, into robotic programs in the device's scripting language. In the case of 3D printers, Ultimaker Cura (Braam 2014) is a popular choice among the so called "slicer" tools, applications used to process digital 3D models into low-level GCode instructions (ISO 6983-1: 2009) for fused deposition modeling.

Programming systems allow creation and customization of robot programs, but are independent from program execution on the target machine; this is the responsibility of a different member. The *robot control system* is the mechanism that implements all the governing instruments to enact a program on the robot, monitor its state at run-time and perform the dynamic adjustments necessary to ensure a faithful execution of the source program. Control systems are typically an integrated part of the controlled device, and regulate its mechanical functions according to the input program. Most industrial robotic arms ship with external physical controllers that incorporate all the control systems to regulate the precise motion of the arm, and are accessible via their vendor interfaces on a handheld pendant. 3D printers, for instance, usually incorporate the control system as part of their firmware, and accessible through their main interfaces.

Robot programming and control are the core components of a traditional robotic design workflow, connecting the principal agents of the system: the *designer* of a robotic task, and the *machine* to successfully execute it. Between them, the *program* becomes the fundamental form of communication, and the vessel for the transfer of *information* and *intention*.

3.2 Offline programming

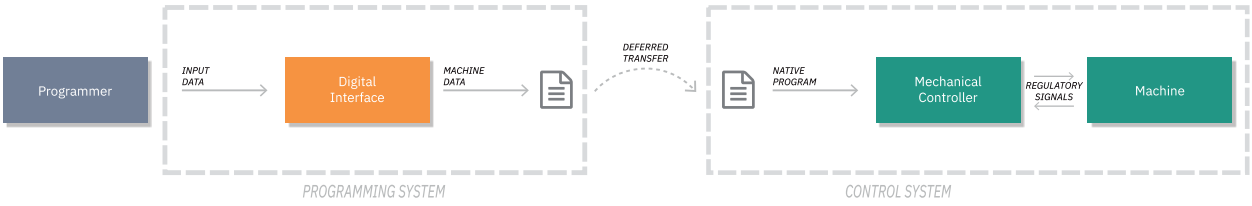


Figure 8 - The split nature of an offline programming environment. The programmer interacts with a digital interface to generate a robot instruction program file. In order to enact that program, the file must be loaded on to a different environment, the control system, who is in charge of regulating its execution. In the following robotic system diagrams, blue represents decision-making agents, orange represents digital mediation systems, and green represents physical mechanical devices to be controlled.

The main outcome of a robot programming environment is a text file that contains a procedure written in the machine's native programming language; running the procedure on the device will result in execution of a task, —hopefully— fulfilling the intended goals of the programmer. In this case however, program execution on the machine is entirely independent from the programming system that generated it. It is usually up to the programmer to load and run the program on the device, with more sophisticated tools eventually providing

some form of network uploading capacity. But whether if the file is manually or automatically transferred to the machine, program execution happens strictly within the control system, and the programming counterpart is completely detached from the device during execution. This model is commonly referred to as *offline programming*, and is schematically described in Figure 8.

Various implications are derived from the interaction model brought forth by the offline paradigm. The divide between programming and control means that program execution is *deferred* from program creation; it may happen seconds, hours or even days later, but at that point, the program has become *static*, and changes to it require further rounds of compilation and subsequent execution. The architecture of this system fosters a distinct *unidirectionality* of data transfer in the workflow: from the programmer to the machine, but not in return. This model also demands elevated *foresight* during the programming stage, as the programmer must be able to anticipate all possible run-time scenarios before program execution, and embed associated decision-making pre-encoded within the program itself. Furthermore, as the program runs in *isolation* on the device, all the structures necessary to incorporate additional inputs during execution must be available within the device itself, in the form of appropriate software and hardware platforms to accommodate those requests. All these factors are responsible for the acute *disconnect* of the creator with their tool present in offline robotic systems.

From an accessibility perspective, this model carries several disadvantages. The split nature of the programming and execution environments makes iteration in robot programming a slow and laborious task. The need to switch between systems in order to try new program implementations results in lengthy cycles of design and testing, and often, unexpected results or errors in the program are difficult to trace on the controller side, given the lack of program debugging tools. In a good case scenario, the programming environment may provide debugging tools and simulation checks for operations that are intrinsic to the machine. But if successful program execution relies on the correct interaction of the robot with its physical environment, then these tools often fail to provide a predictable and accurate representation of it; debugging must happen in the real world, and at real-world time. Offline programming strategies reflect industry standard division of labor where an "engineer" might develop the robotic program, and an "operator" supervises the execution on the shop floor (Peek 2016). But ultimately, the experience of programming robots offline is cumbersome and tedious, and lacks the desired fluidity of a design environment that fosters immediacy, iteration, experimentation, and "thinking-while-doing" (Sennett 2008).

The foresight required in order to implement a successful robot program, paired with the expertise necessary to develop such foresight, pose an additional entry barrier to the world of robotics. As the immediacy of the robotic medium is undermined by tedious testing cycles, domain knowledge is generally acquired through formal training and abstract reasoning, rather than direct interaction with the machine. However, symbolic learning requires significant time and effort, even for those well versed in the conceptual tools that support it; for the rest, such investment is too often a terminal obstacle that inhibits the development of general-purpose applications, creative inquiry and, in particular, curiosity. The tangibility of the robot as an "object-to-think-

with" (Papert 1980, 11) is a fantastic affordance that gets lost whenever its access begins somewhere else outside the medium itself.

When considered from a technical point of view, the offline model is also constraining in many aspects. The unidirectional nature of its architecture makes it fairly easy to create programs that impose deterministic behavior on the robot. This is convenient in situations where the context of the robot is highly predictable, like a manufacturing plant. Nevertheless, wherever an elevated degree of indeterminacy is present, or when the behavior of the program should be driven in response to mutating conditions around the machine, robot programming rapidly escalates in complexity. As the designer of the program becomes detached from its execution, all decision-making must be pre-embedded into, and delegated to, the algorithms within the program itself. Identifying all possible permutations of cause-effect relationships in the behavior of a robot is not a trivial task; accounting for them in anticipation is a daunting one, even for the most seasoned roboticist. Closing the information loop, and adding the programmer —or any other external agent— as active participants in decision-making processes during execution, are possibilities not in the nature of offline programming (Willis et al. 2011).

The hurdles around creating programs that incorporate external inputs go beyond the complexities of algorithm design, and extend also to their technical integration with the machine. Interacting with peripheral devices, incorporating additional decision-making agents or, in general, exchanging information at run-time with systems outside the robot itself, becomes only possible to the extent permitted by the hardware and software configurations of the device. For instance, it is typical for industrial robotic arms to feature an input/output bus for digital/analog signals, and allow for some kind of low-level network communication like TCP/IP. This makes it possible to integrate industrial sensors, and some form of basic monitoring, by those versed in mechanical engineering or computer science. However, adding other agents as active participants of the robotic application, whose nature is not accounted for in the robot control systems, becomes only possible if the programmer has the technical expertise sufficient to develop the necessary cross-platform integrations; yet another technical barrier for the non-experts.

Ultimately, the offline paradigm only widens the traditional divide between thinking and doing (Carpo 2011). The separation between the digital and the physical machine control systems, the deferred nature of the programming workflow, and the disconnect between the parts within an ideally integrated system, makes reconciling the creator with their tools an unattainable endeavor. There is a profound connection between the craftsperson and their tools (Sennett 2008), a direct contact that fosters reflection, experimentation and discovery (Schön 1984), even on digital environments (McCullough 1996). Offline programming pushes the programmer away from the machine, rather than bringing them together into a closer and fertile relationship. Creators are deprived from the tangibility in the making processes, from the benefits of rich physical interactions (Ishii and Ullmer 1997) and from the textility of making (Ingold 2009).

3.3 Enactive robotics

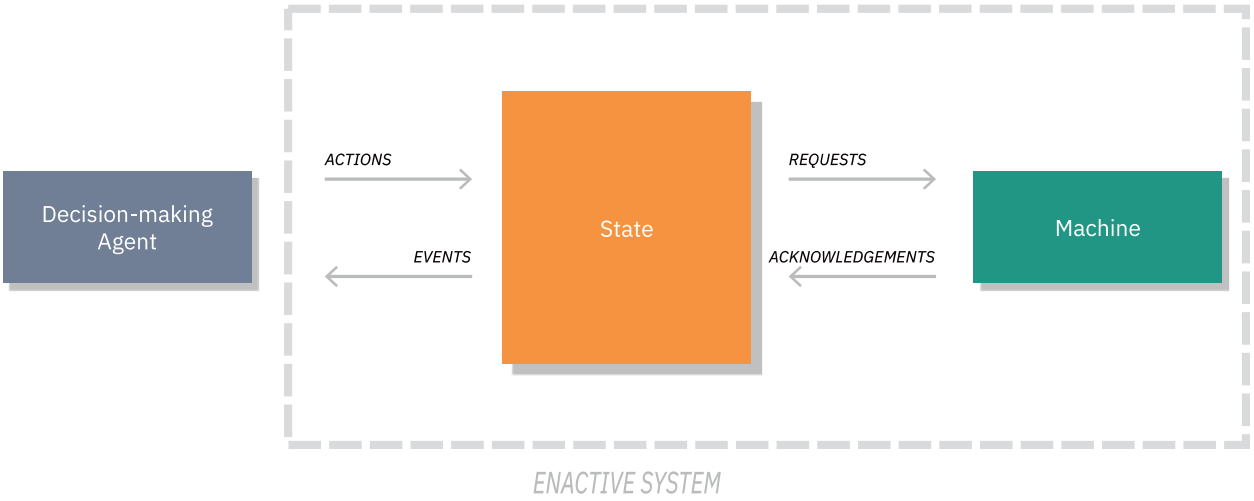


Figure 9 - Enactive robotic systems blur the boundary between programming and control, providing an environment where programs can be enacted *on the system during the design phase. Such integrated, concurrent system allows bi-directional streams of information, and fosters programming styles that are reactive to, rather than prescriptive about, dynamic changes in the robotic system.*

The divide between robot program design and execution fosters a programming style that poses a great entry barrier for the newcomer, constrains the capacity of the system to accommodate or react to external inputs, and detaches the creator from its tool. While this role division may have made sense historically or for industrial applications (Peek 2016), alternative ways of creating machine instruction are possible, which may open up the field of robotics to a larger group of users, applications and interaction designs.

The goal of the *Enactive Robotics* model is to supersede the traditional offline standard in machine programming systems, by systematically incorporating real-time control as the cornerstone of this new programming paradigm. The main proposition of this model is blurring the line between the system used to instruct the robot what to do and the one regulating such execution, integrating them into a single framework where *robot programs can be executed concurrently during the design phase.*

On an enactive robotic system, the programmer interacts with an environment where there is no clear separation between where robotic instructions are created, and where their execution takes place[9]. During program composition, the programmer has the capacity to seamlessly execute individual instructions, partial procedures or complete programs. Immediate feedback can be gathered from concurrent execution, and changes can be made on the fly and further streamed to the machine. As opposed to the deferred duality of the offline model, within a hybrid programming and control system, programmers have the decision-making capacity of designing what the machine should do, and when this should be done; the system is the one in charge of managing the internal mechanisms by which this is made possible. The interaction paradigm enabled

[9] This is to say, while there probably is a distinct separation at the technical implementation level, this separation does not need to be explicitly apparent to the user, or shall they be concerned about it.

by a system that permits program design, decision making and concurrent execution will hereby be referred to as *enaction*, and its relevance will be highlighted over the course of this dissertation.

A fundamental requirement at the core of this model is the *concurrency* of the agents participating in the system. The system must be able to maintain a live connection with the machine, allowing *data exchange at run-time* with the mechanical controller. Similarly, the system must be able to interact with a *decision maker* during execution. This agent is any entity capable of issuing instructions to the system, and concurrently receiving updates from it, which in turn might be used to inform new action requests. As the unidirectionality typical in offline systems is broken, and all parties involved in the system engage in reciprocal information exchange, the master-slave relationship between controller and controlled partially dissolves, and both agents become comparable participants of a truly holistic robotic system (Figure 9). Furthermore, the capacity of the decision maker to make decisions on the fly is enhanced by the

One of the contributions of the enactive model is opening up machine control to a wider variety of controller agents, such as for example, deterministic *algorithms* —just like in traditional offline programming—, fuzzy logic frameworks —such as *machine learning* models— or even *humans* on the loop themselves. In this model, the emphasis shifts from the program as mediator between design intent and execution, to the decision-making agent as the orchestrator of dynamic machine behavior. Additional agents can be made participants, given their capacity to exchange data concurrently with the system: data streams, remote applications, physical controllers, other robots, other humans, etc. Furthermore, the capacity of the decision maker to interact in real time with the system is enabled by the large-scale difference in computational and mechanical run times; this makes it possible for the model to accommodate agents on the loop whose processing speeds are much "slower" than computational time such as, for example, humans[10]. The work of the programmer now becomes architecting the decision-making structures that will regulate the behavior of the interconnected system, whether through simple algorithms, combinations of agents of hybrid nature, or situating themselves personally at the center of the control mechanism.

3.4 Action Model

The formulation of the term *enactive robotics* in this dissertation is inspired by the developmental theories of Jerome Bruner (1966) and the epistemological perspectives of Varela, Rosch and Thompson (2016). As previously discussed in chapter 2, Bruner would propose that human cognitive capacities in children develop in three stages: enactive, iconic and symbolic reasoning. In the enactive stage, children build their understanding of the world through direct interaction with it. The experience acquired by actuation leads to the possibility of recognizing images associated with action, and the generation of their iconic representations.

[10] These time scale differences between the agents involved in an enactive robotics system make the model especially applicable to computer-machine interaction, and therefore, excludes control systems between agents on similar run times, such as computer-computer interaction.

Ultimately, the development of abstract structures of reason allows for the symbolic understanding and communication of those ideas such as, for example, through language.

From an additional phenomenological perspective, Varela et al. propose enaction as a cognitive process that "consists in perpetually guided action" (173), and which "belong[s] to the relational domain of the living body coupled to its environment" (xxv). This notion characterizes a subject's learning process by the creative correlation between their actions on their immediate environment, and their expectations about their effects; in a certain way, it could be understood as an experiential take on Piaget constructivist theories (Gruber and Vonèche 1977).

Both authors situate purposeful and deliberate *action* by the subject at the core of the enactive cognitive process. Interestingly, in the context of this research, their theories highlight some of the problems identified in the traditional offline model. For instance, Bruner's model is in stark contrast with the way robotics is typically taught and accessed (see §1.3 and §3.2), in which the order is drastically inverted: starting with abstract reasoning, ultimately leading to direct manipulation and instruction. Instead, adopting an enactive approach to the teaching and learning of robotics may help the cognitive processes of those not highly trained in the conceptual structures necessary to understand them symbolically. Furthermore, Varela's point of view is especially applicable to the tangible dimension of machine control. A crucial aspect of the model hereby proposed is the capacity to physically interact with a machine during execution, as a vehicle to its understanding. This is an affordance not available in other less substantial media such as, for example, purely computational systems.

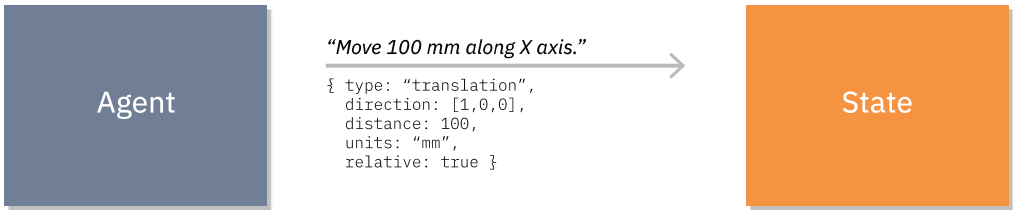


Figure 10 - Actions are the fundamental congnitive units of interaction between decision-making agents and the robotic system. They are self-contained, platform-agnostic, universal, contextual and sequential.

The enactive model proposed in this work is founded on the notion of actions as *cognitive units of interaction* with the robotic system. Within this model, an *action* will be defined as a request from a decision-making agent to change some of the properties of the controlled device. These properties will typically be the ones involved in spatial motion, such as location and orientation, but may also affect internal properties of the machine such as speed, precision, temperature, etc. In a concurrent hybrid programming and control system, actions become the basic units of robotic instruction, acting both as a programming statement and a control block at the same time.

In order to align with the principles of the model, actions should present the following properties:

- *Self-contained*: An action should carry all the necessary data, properties, and modes necessary to enact a particular state change on a machine, e.g. moving it in three-dimensional space, changing the I/O values, or making any modifications to the internal properties of the device. Actions are the smallest units of information required to modify the device and interact with it.
- *Platform-agnostic*: Actions are completely independent of the device they are targeting, and contain no information that is unique to it. Additionally, actions incorporate no prescriptions about the details of how an action should be carried out by the machine or the nuances of their implementation.
- *Universal*: As a consequence, there is no specificity towards any particular machine in the nature of an action, and any action can be requested from any device at any given time during machine execution. Therefore, actions become truly universal units of interaction.
- *Contextual*: Their universal and platform-agnostic properties make actions applicable to any device participating in an enactive robotic system. However, the effect of an action on such device will be determined by the nature of the device and its state at run-time, including the very essential capacity of the device to be able to fulfill the request at all. This is probably the most important property of actions, and will be elaborated in the "State Model" section.
- *Sequential*: The concurrent nature of enactive control systems requires actions to be executed in the sequential order in which they were requested. This allows actions to incorporate the cognitive dimension of time, and maintain a direct relationship between program design and control, independent of particular machine implementations.

Under this model, it is important to highlight the nature of an action as a *request* for state changes. The reasons behind this are related to the intrinsic properties of actions in relation to the concurrency of the system, and its enactive approach to cognition. A consequence of the properties described above is that *any action* can be requested from *any device* at *any time* during the execution lifecycle of the system. This is in stark contrast with typical programming environments, where only procedures supported by their members are permitted, with anything outside them yielding errors upon compilation. However, actions are *not guaranteed* to succeed in their execution. *Success* is here understood as the conceptual match between the expectations from the requester about the effect of an action, and the actual result of its execution; the closer they align, the higher the degree of success.

The term request also refers to the possibility of an action to have no effect at all on a machine —the machine cannot perform the action because of its nature or its current state—, or for the execution of the action to be significantly delayed on time —due to the large differences between computational and machine run times. The variable effect an action may have on a machine, and its relation to the expectations behind that action, contribute to the constructivist cognition of the requester through enaction with the system.

The fundamental properties of actions can be illustrated with a simple example. An action requesting motion could be represented by the description "Move 100 mm along the X axis." This action incorporates all the information it is meant to affect in a self-contained manner: a request for motion, the distance, direction and

units. However, it is agnostic to the platform it targets, as it contains no instructions on how to execute this request on the machine end. The universality of this description makes it applicable to almost any machine that performs motion in three dimensions: an industrial robotic arm, a CNC router, a 3D printer, a drone, etc. However, its effect depends on the context of the machine it is targeting: the final position depends on the initial position of the machine upon request, the machine's internal coordinate system, whether if the position is out of reach from the machine, or simply, just have no effect at all, like if requested to an Arduino board. Additionally, when the machine will start moving will depend on when this action was requested, and how many actions were pending execution before it.

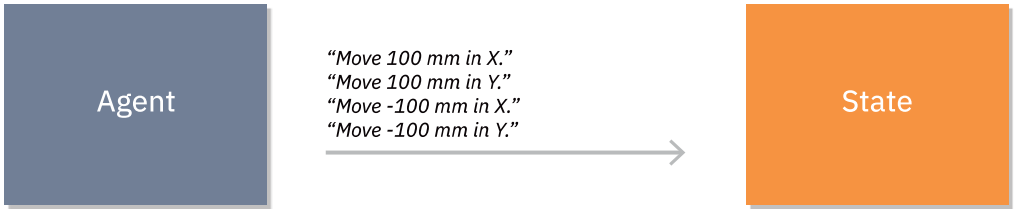


Figure 11 - A "Draw a 100x100 mm square" procedure, composed of a sequence of four unitary actions.

When designing actions for an enactive system, and in order to align with the cognitive goals of this model, it is advisable to conceive them as *high-level representations of low-level requests*. For instance, "Move 100 mm along the X axis" is a good example of a high-level description for a simple concept, one that would be conceptually difficult to break down into subparts, and which is typically ingrained natively in the way translational machines work. However, a "Draw a 100 by 100 mm square" action would be a high-level description for a high-level task that leaves too many open-ended questions to the implementation: how is the square oriented in space? Is it traced counter or clockwise? Is there an actual tool on the device tracing marks on a material? The same task would be much more adequately defined by breaking it down into the steps "Move 100 mm in X, move 100 mm in Y, move -100 mm in X, and move -100 mm in Y" which, by nature of how actions are designed, is guaranteed to execute sequentially; this small program could conform the internal implementation of a higher-level procedure "Draw square" defined by the programmer. The enactive model fosters the understanding of a medium by initially interacting with it via the most basic units of cognition —actions—, and eventually building up towards the complexity of the structures that represent them symbolically —programs.

3.5 State Model

The contextual nature of actions makes their effect on a machine dependent on the properties of that machine at the time the request is fulfilled; in a sense, this concept is akin to the enactive idea of a subject's perception led by actuation on their changing environment. The cognitive tie between an action and its situational effect is key to the understanding of a medium enacted by the subject, and is the guiding principle at the core of the model proposed in this work.

The enactive robotics paradigm is built around the centrality of the *state* model as a *concurrent representation of the changing properties* of the devices participating in a system. The state model conforms the digital instantiation of the robotic environment with which a decision maker can interact via actions, and to which any physical device is a surrogate of.

The central position of the state representation within the enactive model serves multiple functions, with the most important one probably being to serve as the *main interface between decision-making agents and mechanical devices*. The state is the main recipient of action requests, and acts as the interpreter of their meaning based on machine nature and properties. The applicability of an action is validated against the potentiality of the target machine to fulfill it, and its contextual effect is determined based on the concurrent state representation of the system. If a particular action is not suitable for execution by a particular machine, or at a given moment at run-time, it will be rejected by the state model. On the contrary, if an action is suitable for execution, the state model will determine its actual concurrent effect on the machine, and manage the low-level details of its implementation. Additionally, if the information provided by the action is not enough for adequate execution, the state model will decide if the action shall be rejected, or if concurrent state information can be used to complement the request.

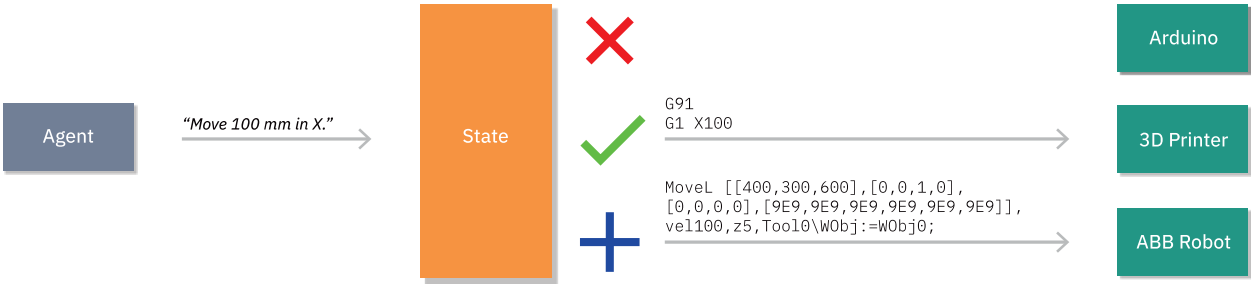


Figure 12 - The role of the State model as the contextual mediator between action requests and the system; a "Move 100 mm in X" action gets rejected for an Arduino board, gets literally translated into GCode for a 3D printer, or would need to be complemented with additional state information — previous position, orientation, speed, etc.— for a RAPID instruction in an ABB robot.

This role can be illustrated with the previous example of a "Move 100 mm along the X axis" action (Figure 12). If such action was requested from an Arduino board connected to the robotic system, the state model would most likely reject it, in the basis of the incapacity of the device to fulfill this request. If the action was requested from a 3D printer, the state model could generate the instructions for a simple relative translation[11]. However, if the action was requested from a robot arm, the state model could either execute the action by computing the final absolute position from current state and by supplying additional spatial orientation information[12], or reject it based on the lack of state data, action ambiguity or out-of-reach conditions.

[11] GCode accepts relative transformations natively.

[12] The robot may not accept relative transformations natively, and hence a relative translation must be internally converted into its final absolute position plus orientation based on concurrent state data.

In order to align with the enactive principles proposed in this work, the state model should present the following properties:

- *Mutability*: The state model should be able to dynamically change in concurrency with execution, in order to maintain a faithful representation of the state of the system and the changes requested by the decision-making agents via actuation.
- *Asynchrony*: Given the large-scale differences between computational and mechanical run times, the state model serves a fundamental role as negotiator between them. As a consequence, interaction with the state model is preeminently asynchronous, and the model is responsible for maintaining accurate dynamic representations of all agents in the system over time, as well as the queued execution of actions in relation to their sequential requests.
- *Mediation*: The state model should incorporate all the mechanisms necessary to act as the mediator between the participants of the robotic system. This includes, but may not be reduced to, state-based translation between high-level descriptions of actions and their low-level machine implementations, communication protocols for data exchange, the regulatory instruments to enact and maintain control over mechanical devices, and notification systems to decision-making agents about relevant state changes during execution.
- *Transparency*: Finally, the state model should be traceable, meaning that any agent should be able to inquire its properties at run-time. The feedback from the environment provided by the state is crucial to the action-effect coupling required by enactive cognitive processes. Additionally, agents may require concurrent state queries for their conditional decision-making processes.

These properties enable the capacity of the state model to provide an enactive environment where programming and execution happen concurrently, given the interaction paradigm provided by actions as units of cognitive state change, and the central management role of the state model as regulator of the control mechanisms in the robotic system. The learning processes that lead to understanding the machine environment via enaction are aided by the feedback provided by the model at run-time, in a much more direct way than with traditional offline paradigms.

From a design perspective, it is the responsibility of the creator of an enactive robotic system to establish the action-effect causality that correlates a suite of original actions to their results when applied to different devices. As manifested by the examples discussed here, the nuances involved in the execution of a high-level action strongly depend on the nature of the machine, its current state and the execution runtime. However, they are ultimately determined by the intent embedded in their design by the creator. Correlating a high-level universal robotic language with the behavior of myriads of different machines is not a trivial task, let alone its technical implementation.

The model presented in this dissertation tries to lay the foundations of a conceptual model to serve as design guidelines for the creation of enactive robotics systems; it will be up to the designer to adapt these principles

according to the nature of the machines to be controlled, and translate them into actual implementations that satisfy the high-level goals of the robotic system at hand[13].

3.6 Summary

In this chapter, *Enactive Robotics* was introduced as a conceptual model for the design of concurrent robot control systems. The model is based on the following key principles:

- The capacity of a decision-making agent to interact concurrently with the system via actions.
- The nature of actions as self-contained, platform-agnostic, universal and contextual requests to change the state of the system, and their role as cognitive units of interaction.
- The central role of a state representation of the system as the mediator between actions and their effect on the actual mechanical device.

In chapter 4, a series of sample technical implementations of this model will be presented, designed for different types of users and frameworks.

The claimed benefits of this model will be discussed in chapters 0 and 0, where a series of case studies and a controlled user test will be presented.

[13] It is important to note that, given the inherent uncertainty about the effect of an action on a particular device, and the safety hazards that may arise form humans controlling machines in real time, it will also be the responsibility of the designer to implement the appropriate security measures to ensure the safety of the system and the people around it.

4 Implementations

Enactive Robotics, as described in this dissertation, constitutes a conceptual model geared towards the design of robot control systems. The model provides a set of integrated guidelines for the creation of robotic frameworks that significantly reduce the high entry barrier typical in standard robot environments, and enable a deeper platform for the creation of interactive machine programs. The model can be useful for designers, engineers, computer scientists or anyone interested in the development of robot control frameworks that foster these properties.

The principles described in the enactive robotics model are in nature abstract, high-level and platform-agnostic. These principles are built to aid the cognitive aspects behind the ways humans understand machine control, and to break away from a long-standing, constraining tradition on how to perform it. As such, these guidelines are applicable to the design of machine control systems for a wide variety of robotic devices.

However, the model does not explicitly provide any details on *how* to implement these principles. There are countless types of programmable machines, each one with different native languages, regulatory mechanisms, communication protocols and nuances of how to interface with them; proposing a one-size-fits-all technical solution might be an unrealistic endeavor. Instead, the model is purposely generic, open to interpretation depending on the nature of the machines that it is going to be adapted for. It will be developer's responsibility to design a high-level application programming interface (API) of suitable actions, conceive the technical architecture of the state representation, and implement the adequate control mechanisms to adhere to the principles hereby described.

In this section, two technical implementations of the *Enactive Robotics* model are presented. The first example is a code library named *Machina.NET* written in C# for the .NET framework. This project implements the full architecture of the action-state model, and is geared towards power users. The second example is an *ecosystem of interfaces*, either built on top of the .NET library or as modular parts of an interdependent system, and are geared towards entry-level users. Their design tries to maximize features that are inspired by the motivations behind the enactive model: universality, intuitiveness, immediacy, cross-platform compatibility, extensibility and education potential. The samples are proposed here to help illustrate possible adaptations of the

conceptual model to a concrete suite of applications, in this case geared towards a specific family of programmable machines: six-axis industrial robotic arms. Both implementations are part of the larger open-source project *Robot Ex Machina* (García del Castillo 2016), or "robot from the machine," named as an allegory for this project's spirit of infusing agency and responsiveness into otherwise inanimate machines.

4.1 Machina.NET

The first implementation presented in this document is named *Machina.NET* (García del Castillo 2019). The project is a library of classes, methods and utility functions designed following the principles of the enactive robotics model.

The library is written in the C# language for the .NET framework. This environment was chosen for multiple reasons: seamless integration with common CAD/CAM applications, powerful and fast architecture, integration with lower-level systems, etc. This project is particularly suitable for users willing to write Windows-based applications that interface with the robot in real-time, or use it at the core of other applications that allow scripting in C#[14].

4.1.1 Hello Robot

```
using System;
using Machina;

namespace Sample
{
    class MachinaSample
    {
        static void Main(string[] args)
        {
            Robot arm = Robot.Create("HelloRobot", "ABB");

            arm.ControlMode("online");
            arm.Connect("192.168.125.1", 7000);

            arm.Message("Hello Robot!");
            arm.SpeedTo(100);
            arm.MoveTo(400, 300, 500);
            arm.Rotate(0, 1, 0, -90);
            arm.Move(0, 0, 250);
            arm.Wait(2000);
            arm.AxesTo(0, 0, 0, 0, 90, 0);

            Console.WriteLine("Press any key to finish this program");
            Console.ReadKey();

            arm.Disconnect();
        }
```

[14] For example, DynamoBIM by Autodesk, Grasshopper3D by McNeel, or Unity by Unity Technologies.

```
    }
}
```

Figure 13 - A sample "Hello Robot" program written in Machina.NET.

A simple application in C# built with Machina.NET could look like (Figure 13). In this example, the sequence of actions is as follows:

- Define a new instance of a `Robot` object, giving it a name, and defining its brand.
- Connect to a physical robot at given `IP` and `Port` values.
- Display the `"Hello Robot!"` message on the device's display.
- Set the internal linear speed value to `100` mm/s.
- Move to Cartesian XYZ coordinates `(400, 300, 500)` in the robot's reference frame.
- Rotate `-90` degrees around the unitary positive Y vector `(0, 1, 0)`.
- Move `250` mm in positive Z.
- Wait for `2000` milliseconds.
- Set the rotational values of the robot's six axes to `(0, 0, 0, 0, 90, 0)` degrees respectively.
- Halt program execution by requesting user input, giving time to the robot to complete these actions.
- Disconnect from the controller before leaving the program.

Figure 14 - The four main motion actions of the "Hello Robot" sample program executed on an ABB IRB 1200 robot - `MoveTo(400, 300, 500)`, `Rotate(0, 1, 0, -90)`, `Move(0, 0, 250)` and `AxesTo(0, 0, 0, 0, 90, 0)`.

The result of the execution of this program on an ABB IRB 1200 robot is illustrated in (Figure 14).

4.1.2 Syntax

Machina.NET is designed to program robots that perform motion in three-dimensional space. It features a high-level API of instructions to describe kinematic transformations, composed of simple English verbs that denote calls to action; this API conform the *enactive Action model*, which will be described in detail in §4.1.3. Some of its syntactical flavor and state-based settings are influenced by the Processing programming language (Reas and Fry 2007).

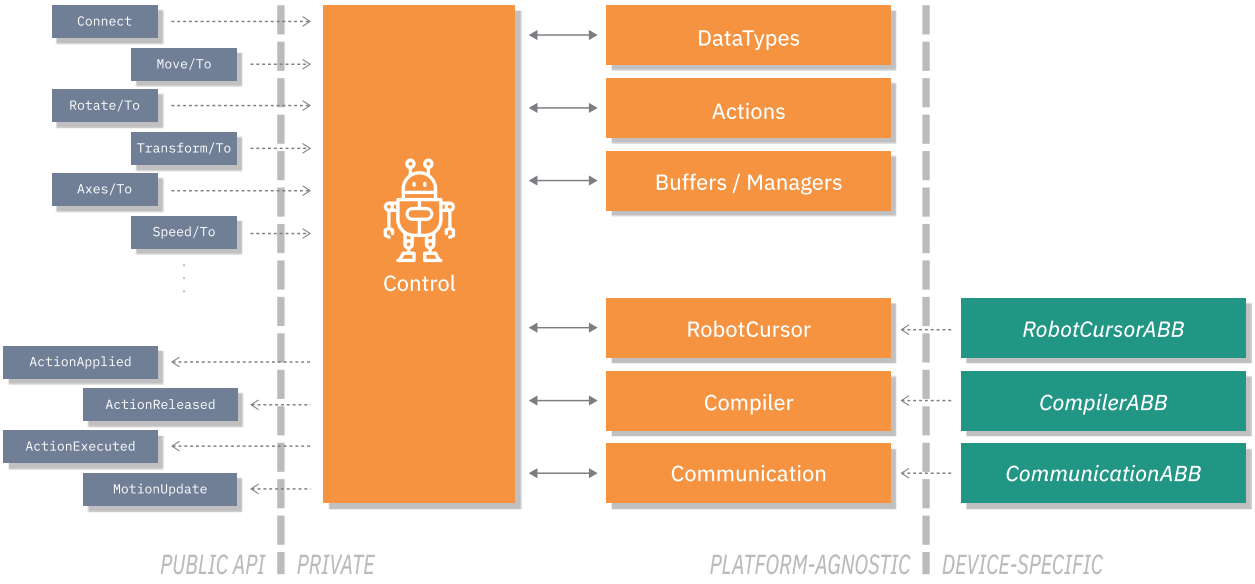


Figure 15 - Overall architecture of the Machina.NET library.

The `Robot` class is the main public interface with the library. It features methods such as `Move`, `Rotate` and `Transform` to request kinematic transformations, `Speed` and `Precision` to request changes in motion properties, or `Connect` and `ControlMode` as management methods. A program can be created by the simple concatenation of a sequence of actions, and running it will result in either the buffered execution of those actions on the connected device, or offline compilation into the device's native language.

Where applicable, most of the API methods have two syntactical flavors: the function name with the `To` suffix, and the plain version. This denotes the difference between *absolute* and *relative* action requests respectively. Relative actions build on top of the current state of the robot, while absolute actions set those values independently of former states. For example, a `MoveTo(400, 300, 500)` call will set the position of the robot's Tool Center Point (TCP) to XYZ `(400, 300, 500)` mm, regardless of its previous position. If followed by a relative `Move(0, 0, 250)` call, the resulting absolute position of the robot would be `(400, 300, 750)`. This applies to TCP location, orientation, and robot axes values, but also to internal robot motion properties such as speed, acceleration, precision, analog out values, etc.

A breakdown of the most relevant methods in the `Robot` class can be found in Table 1 to Table 3.

Instruction	Description	Related Action
`Move/To`	Change the position of the TCP maintaining its orientation.	`ActionTranslation`
`Rotate/To`	Change the orientation of the TCP maintaining it position.	`ActionRotation`
`Transform/To`	Change the position and orientation of the TCP.	`ActionTransformation`
`Axes/To`	Change the rotational values of the robot's axes.	`ActionAxes`
`ExternalAxes/To`	Change the values of one the external axis attached to the robot.	`ActionExternalAxis`
`Wait`	Halt program execution for an amount of time.	`ActionWait`
`DefineTool`	Define `Tool` properties on the robot.	`ActionDefineTool`
`Attach`	Attach a `Tool` to the robot's flange.	`ActionAttach`
`Detach`	Detach all tools from the robot.	`ActionDetach`
`WriteDigital`	Writes a value to a digital out.	`ActionIODigital`
`WriteAnalog`	Writes a value to an analog out.	`ActionIOAnalog`
`Extrude`	Turns extrusion on/off in 3D printers.	`ActionExtrude`
`Message`	Display a message on the device's screen.	`ActionMessage`
`Comment`	Insert a custom comment on a compiled program.	`ActionComment`
`CustomCode`	Insert a custom line of code on a compiled program.	`ActionCustomCode`
`Do`	Applies an `Action` object to the `Robot`.	N/A

Table 1 - Main motion control actions in Machina.NET.

Instruction	Description	Related Action
`MotionMode`	Set the motion type for future motion Actions, like `linear` or `joint`.	`ActionMotion`
`Speed/To`	Change the speed value new Actions will be executed at.	`ActionSpeed`
`Acceleration/To`	Change the acceleration value new Actions will be executed at.	`ActionAcceleration`
`Precision/To`	Change the smoothing radius value new Actions will be executed at.	`ActionPrecision`
`Temperature/To`	Change the working temperature of one of the device's parts.	`ActionTemperature`
`ExtrusionRate/To`	Change the extrusion rate of filament for 3D printers.	`ActionExtrusionRate`
`PushSettings`	Buffers current state settings.	`ActionPushPop`
`PopSettings`	Reverts the settings to the previous state buffered by `PushSettings`.	`ActionPushPop`

Table 2 - Main settings actions in Machina.NET. *Note that all state changes caused by these actions can be buffered and reverted with the use of `PushSettings` and `PopSettings`.*

Instruction	Description	Related Action
`Robot.Create`	Create a new instance of a `Robot` object.	`N/A`
`ControlMode`	Sets the control type the robot will operate under, like `offline` or `online`.	`N/A`
`ConnectionManager`	Defines who is responsible for setting up the controller for connection, Machina or the `user`.	`N/A`
`Connect`	Connects to a remote controller.	`N/A`
`Compile`	Create a program in the device's native language with all the buffered Actions.	`N/A`

Table 3 - Main management methods in Machina.NET. *These instructions have no actions associated to them, as they are related to setup rather than execution.*

4.1.3 Action Model

The internal programming and control model of *Machina.NET* is based on the atomic form of `Action` objects. In line with the enactive model, an `Action` is defined as a change in some of the properties that define the state of a robot. For example, this can mean to perform some kind of spatial motion —`Move`, `Rotate`—, change the execution properties of forthcoming actions —`Speed`, `MotionMode`—, hold execution for a certain amount of time —`Wait`—, change the IO values —`WriteDigital`, `WriteAnalog`—, etc.

Table 1 to Table 3 denote a loose correlation between `Robot` methods and `Action` types. This is not coincidental, as most API calls are basically thin wrappers that create an `Action` object internally and issue a request to the core `Control` class to execute that action. This pattern presents a clear mental model to the user, who can easily relate the statement `arm.Move(0, 0, 250)` to the meaning "ask the robot to move up 250 mm," and doesn't need to care about the internal representation of such actions. It also helps maintain the modularity of the library and makes extending its functionality much simpler.

Action objects follow the fundamental properties described in the enactive model. For example, actions are platform-agnostic; they are objects which store values for the state properties they are meant to modify, but do not need information on their target machine in order to exist —even if they are created through the `Robot` class. This means that programs are created internally as lists of `Action` objects, but these actions have no real meaning until applied to a particular device. Depending on its nature, such device may or may not be able to accommodate those changes; it is up to the developer who extends the library for a new machine type to decide which actions have effects on it and what their effects are. This means that calling the `bot.ExtrusionRateTo(100)` method on a non 3D printer is a valid operation, but will have no effect on the machine when executed online or compiled offline, beyond a warning message. This allows to maintain the cross-compatibility of the same API between different types of robots.

Machina.NET can be extended with new actions by inheriting from the main `Action` class, and adding corresponding methods to the `Robot` class. Child actions must implement the `ToString()` method with a human-readable representation of the effect of the action; this helps the user understand the abstractions behind them. Child actions must also implement the `ToInstruction()` method to return a string mirroring the necessary API call to create such `Action`. An *instruction* is therefore defined as the string literal representing a parameterized request for the execution of an action. This is extremely useful as a form of serialization, and provides a way to marshal actions when transferring programs between different implementations of the Robot Ex Machina framework, either through different programming languages or communication protocols. The standard represented by the abstract syntax of the instruction literals, de-serializable into action requests by any member of the Robot Ex Machina framework, will be referred to as the *Machina Common Language*. Figure 16 and Figure 17 show examples of this behavior for the action in our `Hello Robot` sample program.

```
Display message "Hello Robot!"
Set TCP speed to 100 mm/s
Move to [400, 300, 500] mm
Rotate -90 deg around [0, 1, 0]
Move [0, 0, 250] mm
Wait 2000 ms
Set joint rotations to [0, 0, 0, 0, 90, 0] deg
```

Figure 16 - Actions in the "Hello Robot" example stringified to a human-readable program.

```
Message("Hello Robot!")
SpeedTo(100)
MoveTo(400,300,500)
Rotate(0,1,0,-90)
Move(0,0,250)
Wait(2000)
AxesTo(0,0,0,0,90,0)
```

Figure 17 - Actions in the "Hello Robot" example serialized into instruction calls, or the Machina Common Language.

4.1.4 Control Modes

Machina.NET can be used for *offline* programming. In this scenario, the library works by buffering all actions into a program, which can then be compiled into the device's native language, manually loaded and ran on it. This is very similar to the typical way Arduino boards are programmed (Mellis et al. 2007).

Figure 18 shows the `Hello Robot` example rewritten to work in `offline` mode. In this case, `program` contains a string representation of the Machina.NET actions post-processed into ABB's native RAPID language (Figure 19). The same program, compiled in Universal Robots' URScript language ("URScrip Manual" 2018) is shown on Figure 20.

Note how, by default, Machina.NET inserts comments throughout the program with human-readable descriptions of the code. This helps novice users understand the correlation between native instructions and the original Machina.NET actions, facilitating debugging and serving as an entry point to native robot programming. This behavior can be customized through the overloads in `Compile()`.

When working in *online* mode, Machina.NET is intended to be used in control applications running on a host on the same network as the controlled device. In this scenario, the device's controller must be running an instance of a custom *Machina driver module* written in the device's language. Depending on the device, the driver may automatically be deployed by the application, or should be manually set up by the user. The driver module allows the Machina.NET application to exchange information with the device via whichever communication protocol the device accepts. Following the Arduino analogy, this would be akin to interfacing with the board via the Firmata protocol (Steiner 2009). In this sense, the driver module turns the machine controller into a surrogate to the host application, effectively transferring the regulatory and control mechanisms from the device to the system, hence aligning with the principles proposed by the enactive model.

```
Robot arm = Robot.Create("HelloRobot", "ABB");

arm.ControlMode("offline");

arm.Message("Hello Robot!");
arm.SpeedTo(100);
arm.MoveTo(400, 300, 500);
arm.Rotate(0, 1, 0, -90);
arm.Move(0, 0, 250);
arm.Wait(2000);
arm.AxesTo(0, 0, 0, 0, 90, 0);

List<string> program = arm.Compile();
```

Figure 18 - The "Hello Robot" example rewritten for `offline` mode.

```
MODULE HelloRobot_Program

  CONST speeddata vel20 := [20,20,5000,1000];
  CONST speeddata vel100 := [100,20,5000,1000];

  PROC main()
    ConfJ \Off;
    ConfL \Off;

    TPWrite "Hello Robot!";  ! [Display message "Hello Robot!"]
    ! [Set TCP speed to 100 mm/s]
    MoveL [[400, 300, 500], [0, 0, 1, 0], [0,0,0,0], [9E9,9E9,9E9,9E9,9E9,9E9]], vel100, z5,
Tool0\WObj:=WObj0;  ! [Move to [400, 300, 500] mm]
    MoveL [[400, 300, 500], [0.7071, 0, 0.7071, 0], [0,0,0,0], [9E9,9E9,9E9,9E9,9E9,9E9]],
vel100, z5, Tool0\WObj:=WObj0;  ! [Rotate -90 deg around [0, 1, 0]]
    MoveL [[400, 300, 750], [0.7071, 0, 0.7071, 0], [0,0,0,0], [9E9,9E9,9E9,9E9,9E9,9E9]],
vel100, z5, Tool0\WObj:=WObj0;  ! [Move [0, 0, 250] mm]
    WaitTime 2;  ! [Wait 2000 ms]
    MoveAbsJ [[0, 0, 0, 0, 90, 0], [9E9,9E9,9E9,9E9,9E9,9E9]], vel100, z5, Tool0\WObj:=WObj0;  !
[Set joint rotations to [0, 0, 0, 0, 90, 0] deg]

  ENDPROC

ENDMODULE
```

Figure 19 - The "Hello Robot" program in Figure 18 compiled to RAPID language.

```
def HelloRobot_Program():

  popup("Hello Robot!", title="Machina Message", warning=False, error=False)  # [Display message
"Hello Robot!"]
  # [Set TCP speed to 100 mm/s]
  movel(p[0.4,0.3,0.5,0,3.141593,0], a=0.2, v=0.1, r=0.005)  # [Move to [400, 300, 500] mm]
  movel(p[0.4,0.3,0.5,0,1.570796,0], a=0.2, v=0.1, r=0.005)  # [Rotate -90 deg around [0, 1, 0]]
  movel(p[0.4,0.3,0.75,0,1.570796,0], a=0.2, v=0.1, r=0.005)  # [Move [0, 0, 250] mm]
  sleep(2)  # [Wait 2000 ms]
  movej([0,0,0,0,1.570796,0], a=8.726646, v=0.698132, r=0.005)  # [Set joint rotations to [0, 0,
0, 0, 90, 0] deg]

end
```

Figure 20 - The same "Hello Robot" program compiled to URScript.

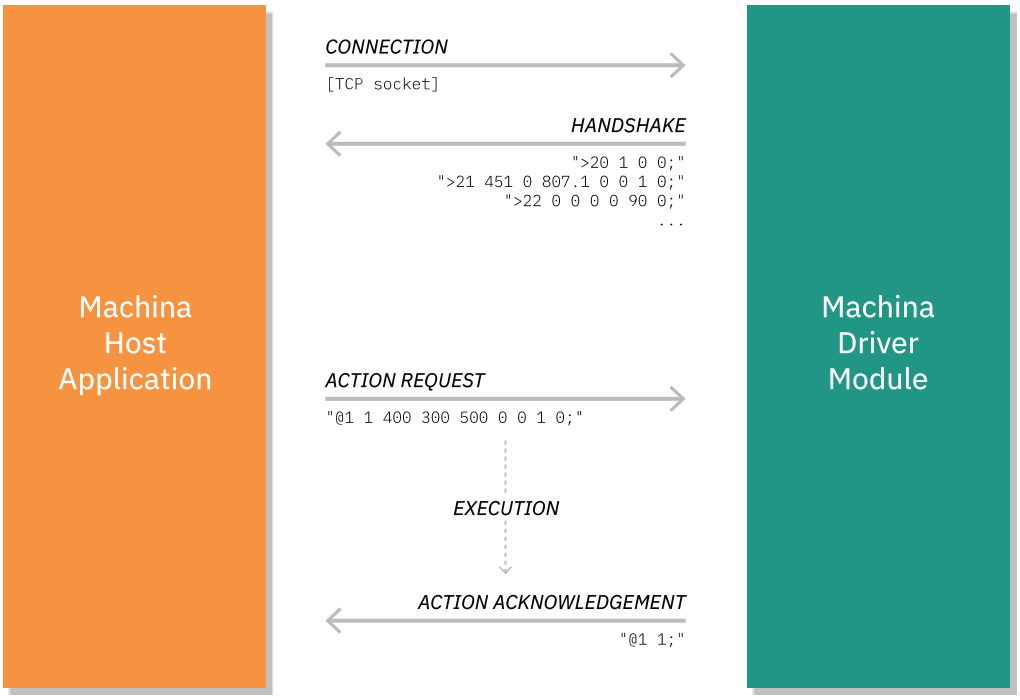


Figure 21 - Sample scheme of a Machina.NET application-driver communication exchange for ABB robots.

Figure 21 describes a high-level representation of a sample communication scheme between a Machina.NET host application and the driver. In this example:

- Communication is established between host and driver through the device's available communication protocol.
- A handshake operation is performed to verify version compatibility and to update the host with initial machine state.
- Host streams action requests to the driver.
- Driver sends acknowledgement messages on action completion.
- Driver streams motion update messages when available.

In the example, communication with ABB devices works by running a driver module that creates a TCP server on the device, accepting incoming socket connections. On successful connection, the client receives string state messages in the format `>21 X Y Z QW QX QY QZ;` and `>22 J1 J2 J3 J4 J5 J6;` with the values for the current Cartesian and joint poses respectively. A request to transform linearly to a Cartesian pose can be sent as `@ID 1 X Y Z QW QX QY QZ;` and, upon completion of this request, the driver will send an acknowledgment message `@ID 1;` including the id value of the completed action. If capable of, the driver may also send state messages periodically, to keep the client updated on the motion state of the device.

However, Machina.NET doesn't impose a particular specification on how drivers for new robots should be implemented. Due to the lack of universality in firmware applications, and the variability of technologies

available on back-end interfaces, different devices may require different communication protocols, connection schemes, and/or message formatting. Developers willing to extend Machina.NET for new machines are welcome to choose how to develop their own driver modules, and which technology stack to implement in order to interface with them. The only requirement is to maintain the updating logic of the robot cursors that represent the asynchronous state of the system (see "State Model" section).

4.1.5 State Model

The *enactive robotics* model proposes the centrality of a *state model* as the mediator between the agents in the robotic system, and as the concurrent representation of their states, in order to aid regulatory control mechanisms and real-time decision-making. As discussed previously, there is a tremendous mismatch between the execution run times of software applications to control hardware devices: a large program may execute in a handful of milliseconds on the digital side, but may need several minutes, if not hours, to complete on its analog counterpart. For this reason, there is an inherent challenge in coordinating the elements of a system whose parts work at such different time scales.

Machina.NET approaches this problem by implementing a system of layered machine state representations named *cursors*. A `Cursor` is a virtual representation of the *state* of the robot, defined as the values of all possible properties a particular device may exhibit at a certain stage of the robot's execution timeline. These may include position, orientation, IO values, temperature, etc., which are particular to the device's capabilities. A simple extension of the `Cursor` object for a six-axis robotic arm could implement properties such as `Position` vector, an `Orientation` quaternion or an array of angular values for the `Axes`, whereas a cursor representing a conventional 3D printer may implement `Position` and `Temperature`, but not require `Orientation`.

Since `Actions` are platform-agnostic, their effect depends on the state of the device at the time of execution. An `Action` is said to be *applied* to a `Cursor` when the cursor representing that state changes its properties based on the nature of the `Action`. An example of this would be the `Position` of the cursor changing from `(400, 300, 500)` to `(400, 300, 750)` after a `Move(0, 0, 250)` action has been applied to it. This model is particularly helpful when controlling machines that do not natively accept motion instructions in relative form, since the absolute values for low-level instructions are always available application-side through the `Cursor` representation of the robot. It is also useful when switching between actions defined in Cartesian and joint coordinates, as the state can be maintained in parallel on both spaces, with translations between them updated through forward and inverse kinematic equations.

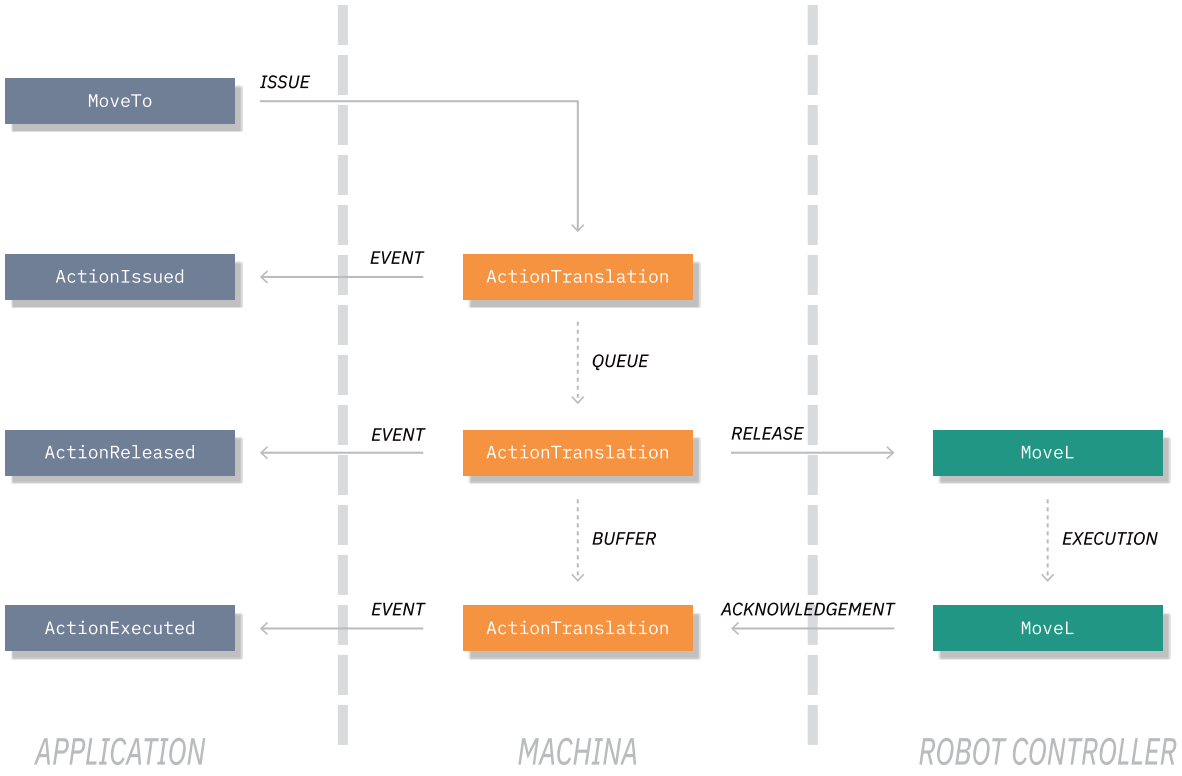


Figure 22 - Life cycle of a Machina Action. `MoveL` *represents a move instruction in the device native's language executed by the driver.*

The asynchronous differences between the host application run time and the robot operation timeline are tracked through a set of cursors representing the different stages of program execution. To better understand this architecture, it will be useful to break down the different stages in the execution cycle of an `Action` (Figure 22):

- An `Action` is *issued* when a request to execute that action has been invoked. This happens typically on most `Robot` API methods such as `MoveTo(400, 300, 500)`. Issued actions are immediately applied to the `IssueCursor`, which therefore maintains a representation of the future state of the robot after program completion, and is the base state on top of which new issued actions are applied.
- Upon issue, Machina.NET queues actions into a first in, first out buffer, an manages the queue according to control mode and robot execution. An `Action` is *released* when it leaves the buffer, and a request to execute that action is sent to the controller. In `offline` mode, all actions are released at once upon `Compile()`. In `online` mode, Machina.NET stages the release of actions to the controller in discrete blocks based on the amount of pending actions on it. Depending on the characteristics of the device, its micro controller may not have enough resources to simultaneously handle communication, instruction parsing and smooth execution of large programs (Wijnen 2016). Blocks of `blockSize` actions are automatically released to the controller whenever less than `pendingCount` actions are pending to be executed, with `blockSize` and `pendingCount` being customizable. This prevents the device from overflowing with data transfers and memory

requirements, and allows for the possibility of on-the-fly modifying or cancelling long programs that have been completely issued but only partially released to the controller. It also facilitates programming styles that are more reactive to the robot execution state. Released actions are immediately applied to the `ReleaseCursor`, which therefore maintains a representation of the state of the robot after execution of all the actions that have been sent to it.

- Once on the device controller, an `Action` is *executed* whenever the changes it represents have been fulfilled by the driver module, and hence the state of the real device reflects those changes. Or in simpler terms, whenever the robot has completed running that `Action`. On successful execution, the host will receive an acknowledgment message from the driver with information about the action that was just executed. Executed actions are immediately applied to the `ExecutionCursor`, which therefore maintains a representation of the state of the real device right after the last completed action. It is noteworthy that, in some instances, and specially with motion actions, program execution on the robot may move on beyond the current instruction even before the robot has fully reached its target position. This is usually the case, for example, when the controller tries to smooth the motion trajectory between forthcoming targets. In these cases, the action is considered executed as well.
- Optionally, some devices have the capacity to run multiple threads and send periodic updates on the state of the robot during execution. In this case, such states can be applied to the `MotionCursor`, which therefore maintains the closest representation to real-time tracking of the state of execution, including intermediate states between actions.

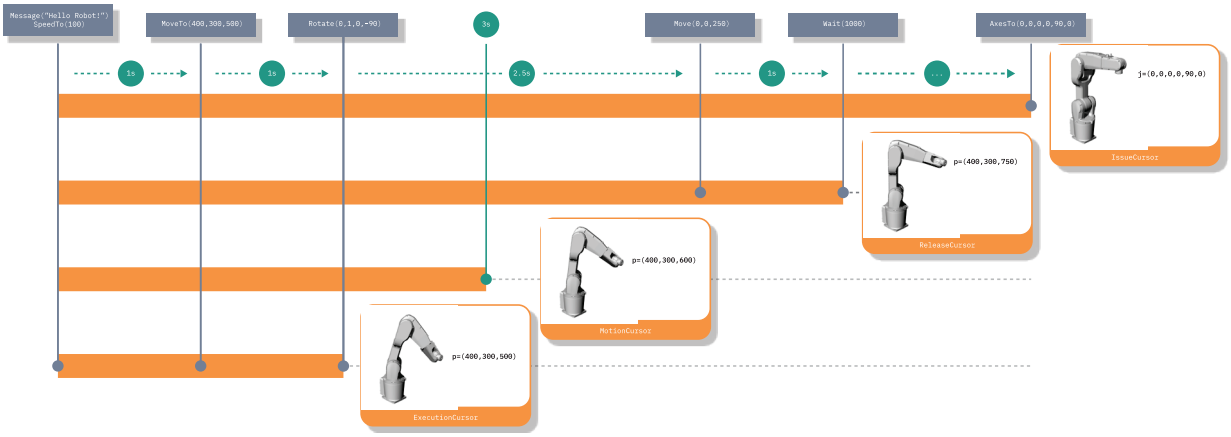


Figure 23 - Layered machine states. This diagram shows the different stages of asynchronous execution and `Cursor` representations for the "Hello Robot" program: from the last known execution state of the machine in `ExecutionCursor`, to the target final state of the `IssueCursor`.

Figure 23 is an example diagram of the different cursor representations of the execution timeline in our initial `Hello Robot` program. Assuming that the initial position of the robot is `(400, 200, 500)`, and that Machina.NET is set to stream to the controller in blocks of six actions, this would be the approximate state of the cursors after three seconds of execution:

- The host application executes in a few milliseconds, and is currently halted waiting for user input. Because all actions in the program have been *issued*, the `IssueCursor` is currently at the last action `AxesTo(0,0,0,0,90,0)`.
- The Machina buffer *releases* actions to the robot controller in blocks of six. The state of the `ReleaseCursor` is updated up until `Wait(2000)`.
- As the robot moves at a linear speed of 100 mm/s, three seconds into the program run time the robot has yet to complete *execution* of `Move(0,0,250)`. Therefore, the `ExecutionCursor` is still on the previous action, `Rotate(0,1,0,-90)`.
- If possible, the device updates Machina with the state of motion at small time intervals. Three seconds into the program, the `MotionCursor` would be approximately at position `(400, 300, 600)`.

4.1.6 Feedback

Machina.NET tries to foster programming styles that are reactive to, rather than prescriptive about, the robot execution state, with interactivity, system input and on-the-fly decision making being at the forefront of real-time robotic applications.

The library exposes a collection of `Events` linked to changes in the `Cursors`. These notify the subscribers of the nature of the changes and other useful information. `ActionIssued`, `ActionReleased` and `ActionExecuted` are raised throughout the different stages of the action execution cycle (see §4.1.5), while `MotionUpdate` is raised anytime real-time information on the state of the robot is received from the controller.

```
using System;
using Machina;

namespace Sample
{
    class MachinaSample
    {
        static void Main(string[] args)
        {
            Machina.Logger.SetLogLevel(3);
            Machina.Logger.WriteLine += Console.WriteLine;

            Robot arm = Robot.Create("InfiniteLoop", "ABB");

            arm.ActionExecuted += {sender, eArgs} =>
            {
              if (eArgs.Pending == 0) Loop(sender as Robot);
            };

            arm.ControlMode("stream");
            arm.Connect("192.168.125.1", 7000);

            Loop(arm);

            Console.WriteLine("Press any key to finish this program");
```

```
            Console.ReadKey();

            arm.Disconnect();
        }

        static void Loop(Robot bot) {
          bot.Move(50, 0, 0);
          bot.Move(0, 50, 0);
          bot.Move(-50, 0, 0);
          bot.Move(0, -50, 0);
        }
    }
}
```

Figure 24 - Whenever the robot has no pending actions left to execute, a new block of actions is issued, generating an infinite motion program.

Figure 24 shows a small program that starts by issuing motion in a horizontal square 50x50 mm loop. Whenever the robot has finished executing an action, `ActionExecuted` is raised. If no actions are pending to be executed, then a new loop is issued, hence creating an infinite loop that can only be interrupted by user input.

Additionally, the library features a full set of `Logger` classes and objects to provide the programmer with relevant information about program composition, un/successful action requests, warnings and errors, etc. Programmers can choose the desired log level —how much information to receive[15]—, manage the received log events or pipe them to standard outs like, for example, via `Console.WriteLine()`.

4.1.7 Other Features

Machina.NET is designed as an introductory platform for non-experts in the field of robotics; in a way, it is like the 21st century Logo turtle of industrial robots. As such, many of the design decisions made during its development were oriented towards fostering simplicity, intuitiveness, safety and getting applications working right away with minimal setup.

The project is fully open source. The aim of this is to open up the field of robotics to curious individuals and power users, either for educational, participation or customization purposes. Furthermore, special consideration is given to thoroughly commenting the source code, to facilitate usage and extensibility of the library.

Additionally, the library has no significant dependencies. It features a full set of custom geometry and robot-related data types, and only references members from the .NET framework. Currently, the only exception is a dependency on the Robot Communication Runtime by ABB Robotics (ABB 2018), used to interface with ABB controllers and automatically run driver modules in the controller. This dependency is scheduled for deprecation in the nearby future.

[15] 0: None, 1: Error, 2: Warning, 3: Info, 4: Verbose, 5: Debug.

Lastly, industrial robots can be dangerous. Robotic actuators are very powerful machines but, for the most part, extremely unaware of their environment; they may not be aware when hitting an object, and continue execution uninterrupted with fatal consequences. Therefore, special attention must be given to the safety of the humans working or interacting with robots. Furthermore, the action-state model in Machina.NET implies that the correct evolution of a program is highly dependent on the successful execution of all the actions in the program's sequence; unexpected consequences may arise from skipping execution of incorrectly formatted or ill-defined actions. Machina.NET incorporates strict checks on correct program definition, and will throw errors and stop program execution upon illegal requests, requiring user acknowledgements and system resets on resume.

4.2 The Robot Ex Machina Ecosystem

Machina.NET is a good example of a project that incorporates, within the same unitary framework, all the elements and system architecture prescribed by the enactive robotics model. However, early user testing (see §0) showed that using the project still posed some entry barriers to non-technical users, due to the following reasons:

- The fundamental prerequisite to accessing the project is for the user to know how to code. And, while there is evidence that an increasing number of members from the creative community are starting to add computer code to their skillsets (Davis and Peters 2013), this is probably still not the norm, and perhaps should not be in any case.
- Additionally, familiarity with the .NET framework, the C# language, building Windows applications and integrating assemblies in other platforms is also strongly desired. However, recent surveys show that C# is not currently among the most popular programming languages for developers[16].
- C# programs are commonly written using the Visual Studio integrated development environment (IDE). This is a professional tool that offers powerful interface for development and error checking, but can be overwhelming and require training for novel users.
- Perhaps most importantly, as any other standard programming language, C# does not provide by default tools to generate or process geometrical entities; users willing to design complex toolpaths, generate path planning or simulate collision checks —all fairly common operations in any CAD/CAM workflow—, must rely on additional external libraries, or write their own geometry processing methods, contributing to the entry barrier for non-technical users that this environment may pose.

The Machina.NET project is probably a great fit for non-expert users who are new to the world of robotics but are tech-savvy, acquaintance with the technology stack described before, and are able to harness the raw power provided by this tool. However, for non-technical users, the abovementioned factors still constitute a

[16] The adoption rate of C# in 2018 was 34.4%, situating it on the 8th position of most popular technologies ("Stack Overflow Survey").

significant entry barrier. The need for a more immediate environment to provide increased accessibility and better CAD/CAM integration possibilities became clear in the early stages of the project.

4.2.1 Machina Bridge

The *Machina Bridge* is the second project presented in this work, and constitutes the cornerstone of the *Robot Ex Machina ecosystem* of robotic applications, for the reasons that will be described further down in this section.

The Bridge is a stand-alone Windows application, built using the Machina.NET library at its core. It is designed to expose the underlying API of actions and utility methods under a dedicated graphical user interface (UI), adhering to the principles and architecture behind the .NET project. This way, the application fosters the immediacy and feedback of the enactive model without requiring coding expertise or the hassle of using complex IDEs.

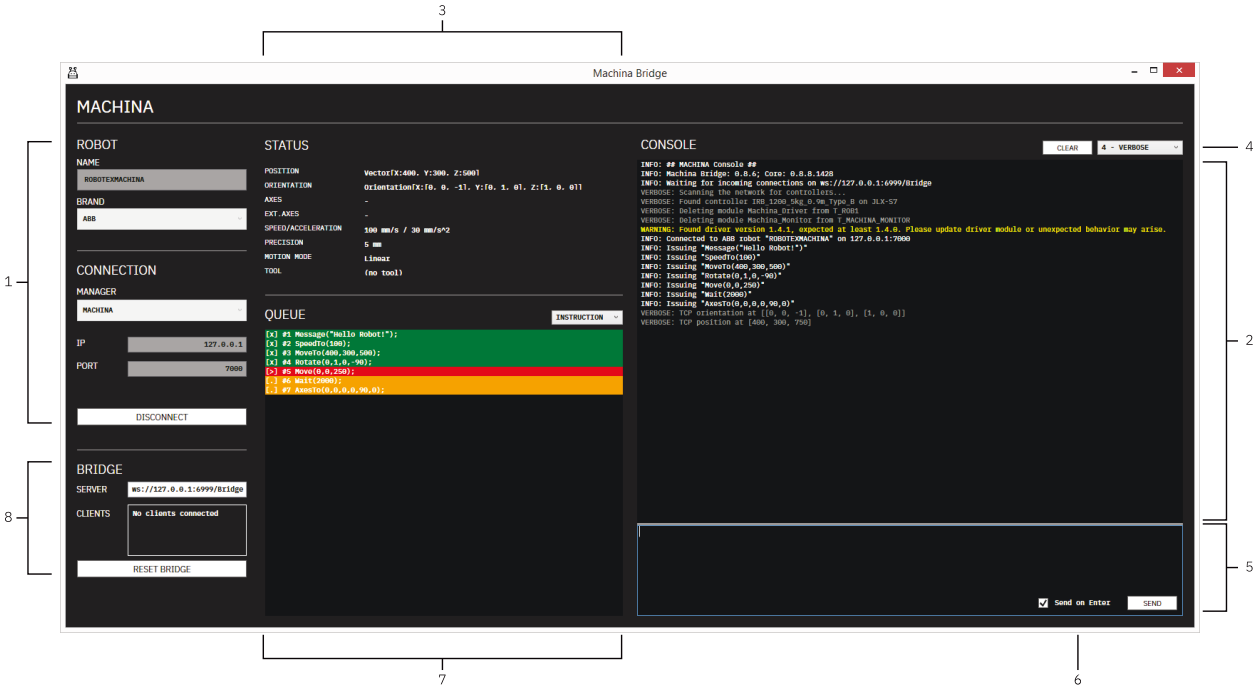


Figure 25 - The Machina Bridge application, featuring: connection panel (1), status properties (3), console window (2) with log level (4), command-line input (5) with send options (6), queue window (7) and WebSocket server options (8).

The application features a panel of options to customize connection settings to the target device (Figure 25). Upon successful connection, the console window displays acknowledgement messages and connection parameters. An additional panel of state properties gives the user feedback on the concurrent state of the system, such as current position, orientation, speed, etc. The amount of information that will be displayed on the console can be customized according to log level. Under the console, a command-line interface (CLI)

window acts as the main input in the system, allowing entry of Machina Common Language instructions (see §4.1.2) such as `Move(0, 0, 250)` or `AxesTo(0, 0, 0, 0, 90, 0)`.

The Bridge is fundamentally designed to work in **online** mode. Therefore, as the user types an instruction on the CLI and presses Enter, the action is immediately requested to the system. If the connection with the machine is live, and the request is valid, the action is immediately buffered and the robot starts executing that action concurrently; if the request was not valid, the console window displays error messages with the reasons behind them. Sequences of multiple actions can be requested simultaneously by disabling "Send on Enter", allowing the composition of small programs on the CLI before requesting them. As soon as actions are requested, the Queue panel displays a list with the buffer of pending actions, and color-codes them based on the real-time execution statuses defined in the .NET library: black if issued, orange if released, red if currently executing, and green if executed (see §4.1.5).

The typical use-case scenario of the Bridge is in fact rather simple: the user opens the application, configures connection parameters, connects to the robot, and starts instructing it what to do. As the user types in an instruction, it gets concurrently executed and the user can see the effect of the action right away; if a block of actions is requested, users can follow program execution in real-time through the queue monitor[17]. The immediacy of this interaction model, as well as the state feedback provided by the UI, are the most prominent features contributing to the enactive character of the Bridge, and their capacity as an effective cognitive tool will be further discussed in chapters 0 and 0.

The Bridge provides an action-state UI for robot control built on top of the .NET library, focusing specifically on fostering immediacy with the mechanical medium, and in line with the enactive robotics model. Nevertheless, it is not designed for advanced features such as simulation, complex program composition or CAD/CAM integration; the interface does not provide tools to deal with these tasks, and they must be handled independently by the user. However, the application can still be of great aid in helping users connect these capabilities with a robot.

One of the most powerful features of the Bridge is its capacity to act as a *gateway to the robot* for external applications. Upon initialization, the Bridge starts an instance of a WebSocket server on the background, which is ready for incoming client connections. The WebSocket protocol was specifically chosen for its capacity to handle bi-directional streams of real-time communication, and its standard availability in most modern programming languages and frameworks[18]. Any application, either locally or remotely, can connect to the Bridge via this protocol, and stream actions to the robot. The client only needs to send string messages to

[17] "The goals of a programming system should be: [...] to enable programmers to see and understand the execution of their programs" (Victor 2012).

[18] "The WebSocket protocol enables interaction between a web client (such as a browser) and a web server with lower overheads, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the client without being first requested by the client, and allowing messages to be passed back and forth while keeping the connection open. In this way, a two-way ongoing conversation can take place between the client and the server" ("WebSocket"). While the protocol was specifically designed for web, it can provide the same communication between applications on the same host machine. Furthermore, WebSocket is a widespread standard that is available natively or as standard packages in most modern programming languages and frameworks.

the Bridge in the Machina Common Language standard —equal to the ones that would be entered in the CLI window (Figure 26)— and those actions will enter the execution buffer if applicable. Additionally, the client will receive string messages in JSON format ("Standard ECMA-404") with notifications about relevant changes in program execution, corresponding to the `Events` described for underlying Machina.NET library (see §4.1.6).

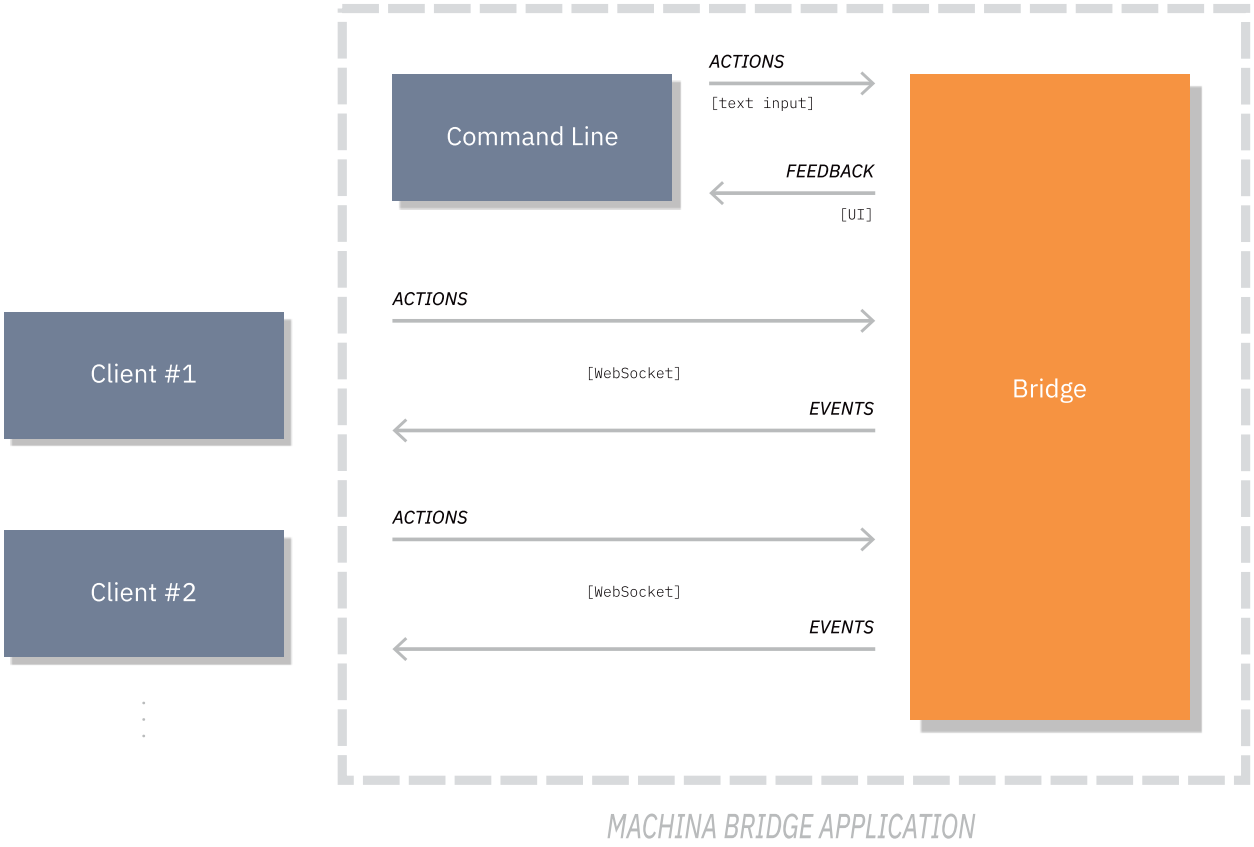


Figure 26 - The Machina Bridge application accepts action inputs from the embedded command-line interface, or from external WebSocket clients, acting effectively as a stand-alone State model for external agents.

When used this way, the Bridge becomes a de facto *stand-alone state model*, acting as the interface between a robot and the actions requested by any other application —hence its name. Users willing to create interconnected cross-platform robotic systems do not need to develop a full technical implementation specific for their environment of choice. Instead, they only need to implement sending action instructions to the Bridge, and handling the reception of event notifications; the Bridge can be relied on as the middleman to the machine for the rest of the system's architecture.

Additionally, the Bridge still provides UI feedback on the state of a robot controlled by an external client: valid incoming actions requests are added to the queue, and dynamic changes in the execution of the robot are reflected on the state panel. Moreover, multiple clients can connect to the Bridge simultaneously and request actions from the connected device, with such actions added to the queue on a first-in first-out basis. This also includes the command-line interface, which can be used to request actions from the robot concurrently with

any other external application. In this sense, the CLI can be conceived as just another client that is embedded in the UI.

The Machina Bridge is a tool designed to provide a simple, self-contained and actionable façade to the underlying architecture of the Machina.NET project. It emphasizes the immediacy and tangibility of real-time robot control by direct interaction with the machine via the command-line interface. It also opens up the model to interaction with any external application able to communicate with it; its role as mediator between the machine and any other system agent situates it at the *center of an ecosystem of applications* designed to interface with it, part of the Robot Ex Machina project.

4.2.2 Programming Languages

The simplicity of the Bridge UI is convenient for rapid setup and direct interaction with the machine, but lacks the complexity to perform advanced programming features such as iteration, conditionals, object-oriented-programming (OOP), etc. The enactive robotics model proposes actions as the main cognitive units of interaction with the robotic system, but does not define higher-level structures to organize them; this is open to the system developer to design.

Additionally, users at a mid-level technical skill may find themselves uneasy with the programming depths of full application development, but increasingly comfortable in smaller-scale scripting languages, simpler program composition or other programming environments. Replicating the full architecture of the .NET library for other frameworks would be a herculean effort, with the additional difficulty of maintaining them all in parallel; any small behavioral change to the main project would need to be replicated to all the siblings in order to maintain their coherence, a task that scales in complexity exponentially.

The Robot Ex Machina project approaches the need for cross-platform compatibility by offering *action models written natively in alternative programming languages*, which *delegate the state model* and system architecture necessary to control the physical devices to the Bridge. This section presents three such implementations: Machina for Processing, Python and JavaScript[19]. Given the widespread adoption of the former framework among the creative coding community (Reas and Fry 2007), and the growing popularity of the latter two languages among professional programmers ("Stack Overflow Survey"), they seemed convenient platforms to extend the functionality of the Machina ecosystem to.

[19] Machina for Python was developed at Autodesk Boston during the Summer of 2018 by Ardavan Bidgoli, Ph.D. candidate at Carnegie Mellon University. Machina for Processing and JavaScript were developed by the author. They can all be found under the main Robot Ex Machina repository (García del Castillo 2016).

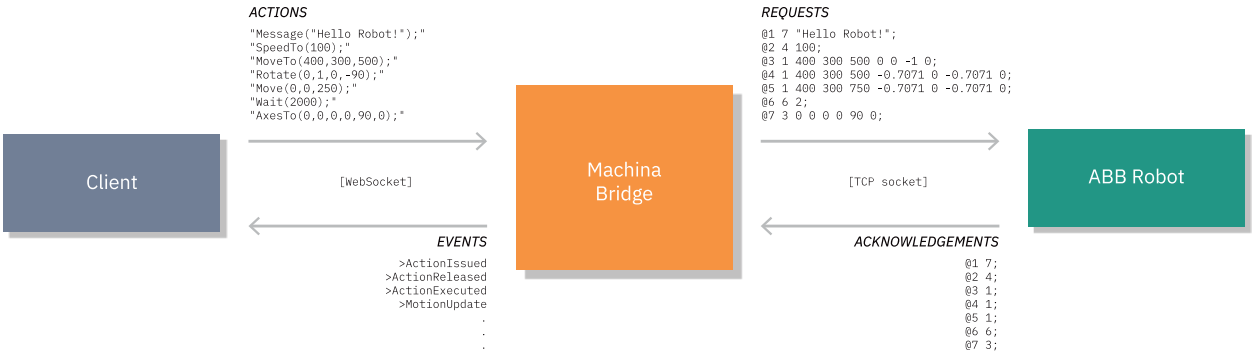


Figure 27 – General architecture of the enactive model via the Machina Bridge: any external agent can connect to the Bridge as a WebSocket client, and send action requests as strings in the Machina Common Language. The Bridge takes the role as the State model, buffering actions and turning them into machine requests when applicable. In this example, the Bridge turns actions into TCP messages for an ABB robot running a Machina Driver Module. The module responds with acknowledgement messages once execution is complete. These, in turn, are translated and notified to the client as asynchronous events.

The architecture of the three projects follows the same pattern: each of them is a thin wrapper around a WebSocket client that provides a *Machina-styled API to communicate with the Bridge* application (Figure 27). For each implementation, this means the following:

- The project is built as a library/package written in the target language.
- The library exposes a main class named `Robot` or similar.
- From this class, an instance of a `Robot` object can be created, passing connection parameters to the Bridge.
- The robot instance is used to establish communication with the Bridge via an internally managed WebSocket client.
- The `Robot` class features a full set of action methods accessible through the instance, mirroring the main Machina API, like for example `arm.MoveTo()` or `bot.Wait()`. Depending on the typing rules in the language, these methods may impose restrictions on the data types they accept as parameters.
- The function of the action methods is to verify the nature of the arguments, and perform their correct parsing into an instruction literal that adheres to the Machina Common Language standard (see §4.1.2). Once a valid string instruction has been internally composed, the object instance sends the message via the WebSocket client to the Bridge, and program execution resumes.
- Additionally, the project should feature some way of triggering responses after incoming messages from the Bridge. Depending on the framework, this might be implemented via event handlers, method overrides, callbacks or similar.

```
import websockets.*;
import machina.*;

WebsocketClient wsc;
MachinaRobot bot;

void setup() {
  wsc = new WebsocketClient(this, "ws://127.0.0.1:6999/Bridge");
  bot = new MachinaRobot(wsc);

  bot.Message("Hello Robot!");
  bot.SpeedTo(100);
  bot.MoveTo(400, 300, 500);
  bot.Rotate(0, 1, 0, -90);
  bot.Move(0, 0, 250);
  bot.Wait(2000);
  bot.AxesTo(0, 0, 0, 0, 90, 0);
}
```

Figure 28 - The "Hello Robot" program in Processing.

```
from machinaRobot import *

def main():
    bot = MachinaRobot()
    bot.message("Hello Robot!")
    bot.speedTo(100)
    bot.moveTo(400,300,500)
    bot.rotate(0,1,0,-90)
    bot.move(0,0,250)
    bot.wait(2000)
    bot.axesTo(0,0,0,0,90,0)

if __name__ == "__main__":
    main()
```

Figure 29 - The "Hello Robot" program in Python.

```
import WsClient from 'WsClient.js';
import Robot from 'machina.js';

let wsc = new WsClient("ws://127.0.0.1:6999/Bridge");
let bot = new Robot(wsc);

function start() {
  bot.Message("Hello Robot!");
  bot.SpeedTo(100);
  bot.MoveTo(400, 300, 500);
  bot.Rotate(0, 1, 0, -90);
  bot.Move(0, 0, 250);
  bot.Wait(2000);
  bot.AxesTo(0, 0, 0, 0, 90, 0);
}

start();
```

Figure 30 - The "Hello Robot" program in JavaScript.

In Figures Figure 28 to Figure 30, the original `Hello Robot` example (see §4.1.1) is showcased written natively for the three frameworks: Processing, Python and JavaScript[20]. As described in the previous sections, actions requested to the Bridge via these programs would enter the Bridge pipeline, appear on the queue panel, trigger immediate execution on the robot, and notifications about relevant execution stages will be received back from the Bridge. The state panel would change dynamically during execution, and on-the-fly intervention would be made possible via the command-line interface.

The architecture behind these projects can serve as an example of how the action-state model encapsulated in the .NET library, and exposed as a stand-alone application in the Bridge, can be extended to virtually any programming language. This architecture has the advantage that most of the technical heavy lifting is performed by the .NET core library, and properly maintaining the client only requires ensuring up-to-date consistency with the Machina Common Language. Even if this was not the case, the enactive nature of the system would result in obsolete actions triggering deprecation warnings, or completely invalid ones being rejected by the system.

Users willing to extend the Robot Ex Machina ecosystem to their own platform of choice could adhere to these guidelines, and harness the power of the framework with little extra effort. The flavor of the API might differ from other projects, as different programming languages promote different design conventions; consistency is desired in any case.

4.2.3 Visual Programming Languages

The recent popularity of visual programming languages (VPL) for parametric modeling (Davis 2013) has led to an explosion of tools that have allowed the proliferation of new forms of expression, backed by algorithmic and computational thinking (Aish 2009). Fundamental to this new culture have been, for example, the Grasshopper3D plugin for McNeel's Rhinoceros (Rutten 2009), and DynamoBIM for Autodesk's Revit (Autodesk 2018). Their capacity to establish parametric workflows for geometry processing make them suitable for the generation of highly complex form —to an extent that some authors claim it as a new style (Schumacher 2009)—, but also for all the geometry processing typically required in complex CAD/CAM workflows: slicing for 3D printing, toolpath generation for milling or motion planning for robotics. This is the main reason why so many tools have evolved in the last few years to address the missing integration of these frameworks with robotic devices, and fuel recent experimentation in the field (see §1.4). However, most of these tools are closed-source, platform-specific, and most importantly, perpetuate the offline programming paradigm; the opportunity was available to contribute a new interaction model to the already busy universe of robotic tools

[20] The example in Figure 30 is designed to run as a Node.js application.

for parametric modeling environments, while at the same time, integrating their geometry processing power into the concurrent control capacities of the Machina framework.

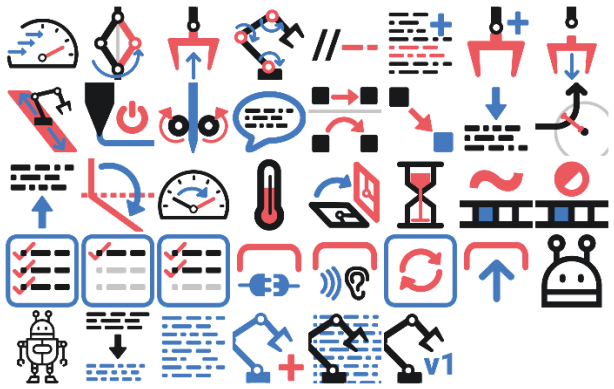


Figure 31 - Machina actions and events as icons for visual programming languages.

Two implementations of the Machina architecture for VPLs are finally presented in this section: Machina for Grasshopper3D and Machina for DynamoBIM (García del Castillo 2016). Both projects are built on top of the Machina.NET library at the core, and distributed as open-source plugins/packages that integrate into their respective hosts. While it could be argued that both Grasshopper3D and DynamoBIM support very different platforms and needs of their respective communities, they will be here discussed under the same category, given that the programming paradigms they provide and the interaction models they support are significantly aligned in spirit.

The main goal of the projects is to provide access to the complete set of functionalities in the underlying Machina.NET project, while at the same time, allowing the possibility of a *full action-state concurrent control* paradigm inside platforms typically designed for directed graph control rather than iterative execution loops.

Figure 32 - The Machina API for DynamoBIM.

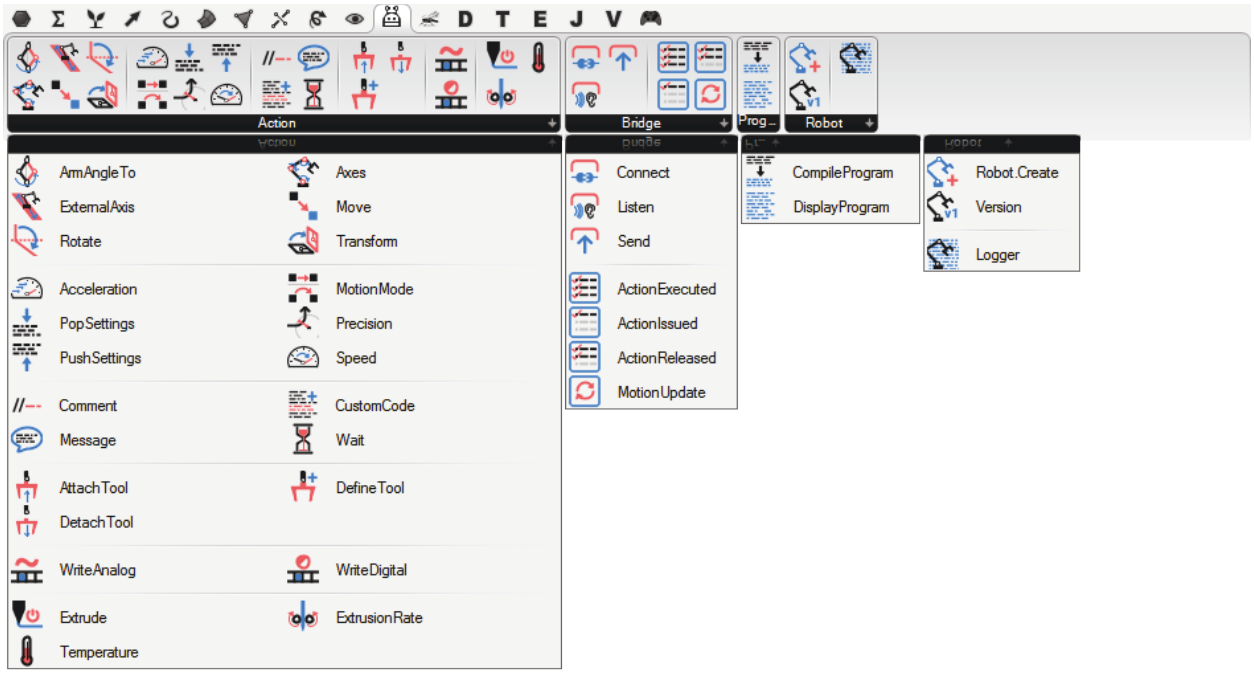


Figure 33 - The Machina API for Grasshopper3D.

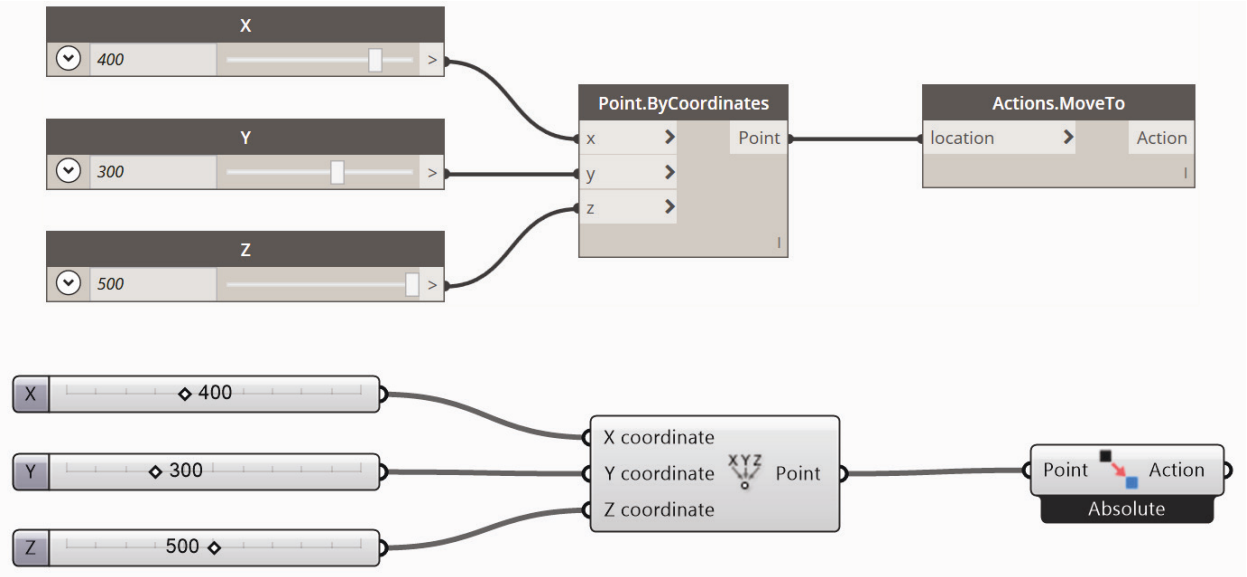


Figure 34 - A `MoveTo` action created from a `Point` object in DynamoBIM (above) and Grasshopper3D (below).

The projects feature a main category named `Action`, which contains a suite of nodes/components representing all possible actions in the Machina API (Figure 32 and Figure 33). These nodes are present in both relative and absolute flavors —where applicable—, and constitute the basic building blocks of program composition. The convenience of using native geometrical objects in the typical workflows provided by VPL applications is the source for the main API difference between these Machina implementations and the core library: while the latter uses primitive parameters for action instantiation —like doubles, integers or booleans—, the former takes geometrical entities from the host application instead as input parameters —like points, vectors, planes, etc. (Figure 34) The nodes take care internally of breaking these entities down and performing the necessary conversions into primitive parameters.

Program composition in these platforms is reduced to the sequential aggregation of actions in ordered lists via the data management tools provided by the platforms (Figure 35). In order to aid debugging efforts, actions are converted by default to human-readable string descriptions of their function (Figure 36). Once full programs are composed, the `Robot` and `Program` categories provide nodes to define specific robot instantiations and compile programs for those target platforms. Offline native code can be generated for any device supported by the .NET core, such as ABB, KUKA, Universal Robots or 3D printers (Figure 37). Additionally, programs can be "compiled" into the Machina Common Language (Figure 38). This serves two main purposes: help the cross-platform integration of the applications in the Machina ecosystem —the Machina-style instructions can be copy-pasted in the Bridge and executed—, and educate users in non-visual representations of those programs —hopefully leading to regular users eventually becoming power ones[21].

However, implementing the full concurrent action-state model within the DynamoBIM/Grasshopper3D environments would be against their very own nature. The programming paradigm they provide is founded on the directed node graph model, and execution is bound to a single cycle of cascading updates; maintaining a live iterative execution cycle that would allow bi-directional data exchange is not contemplated as part of how these tools work.

[21] "The performance of a user interface when operated by an experienced user is ultimately more important than in its use by beginners, as users should remain beginners for only a short period" (Wake 1992, 135).

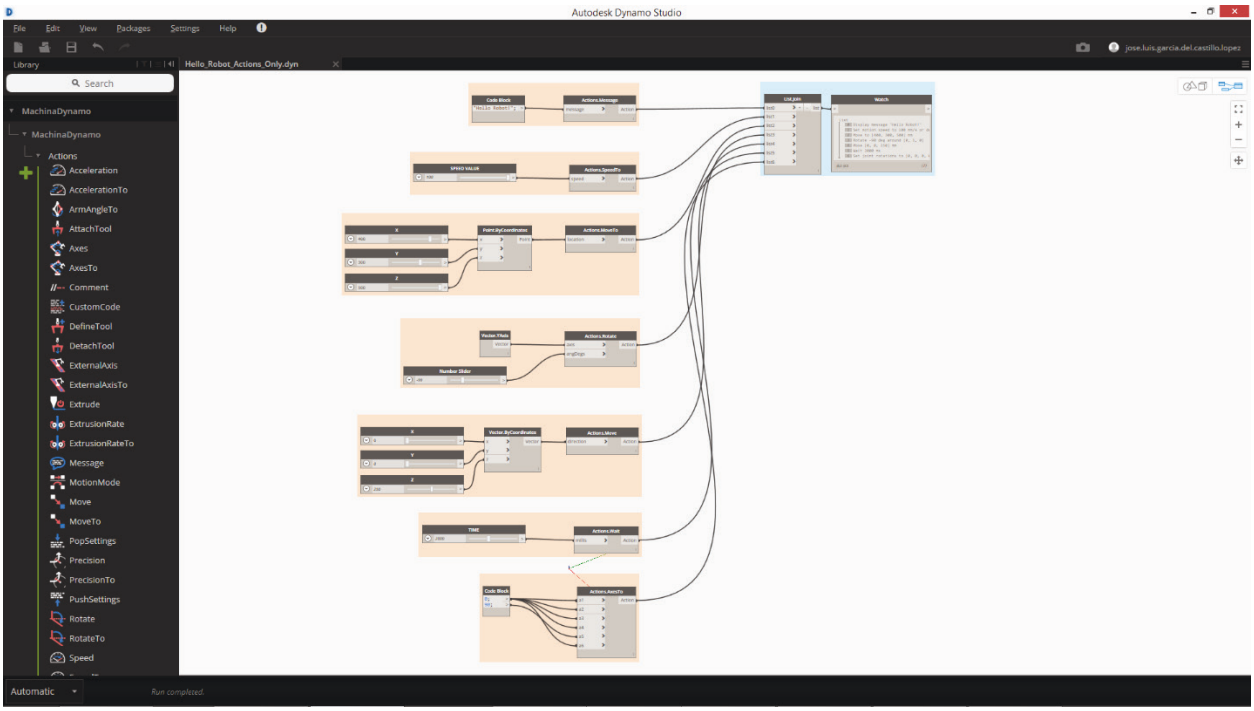


Figure 35 - The "Hello Robot" program in DynamoBIM.

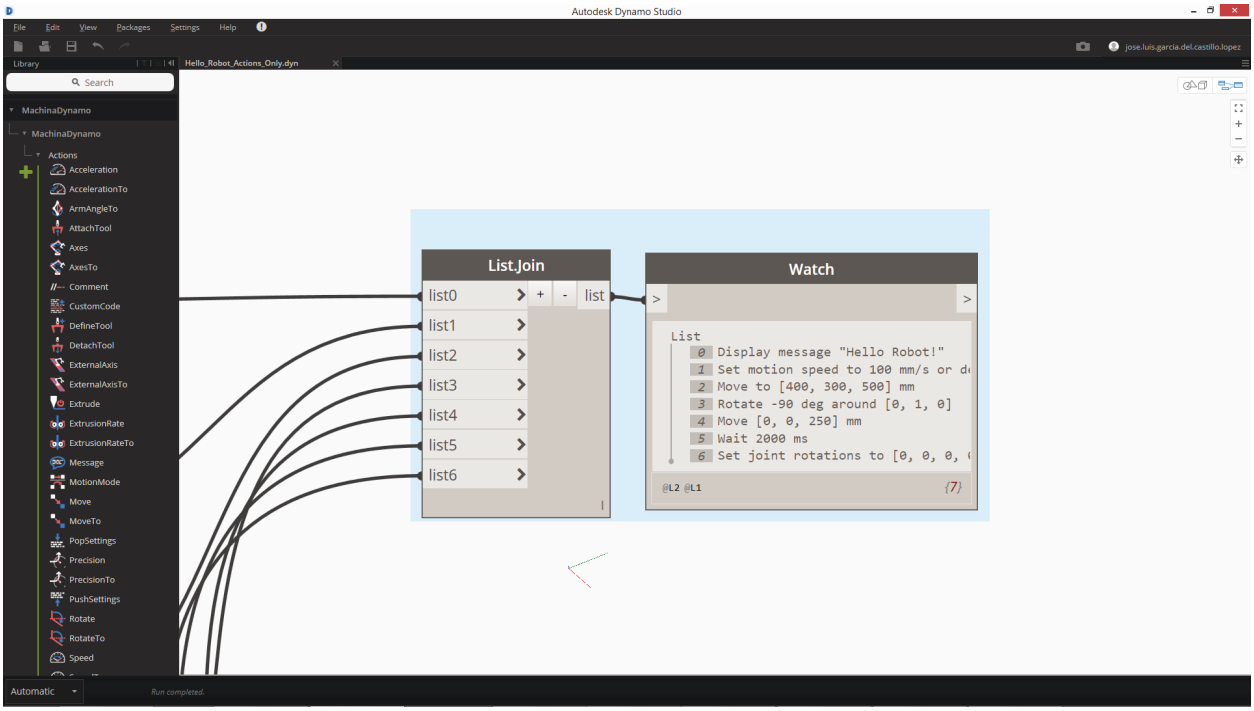
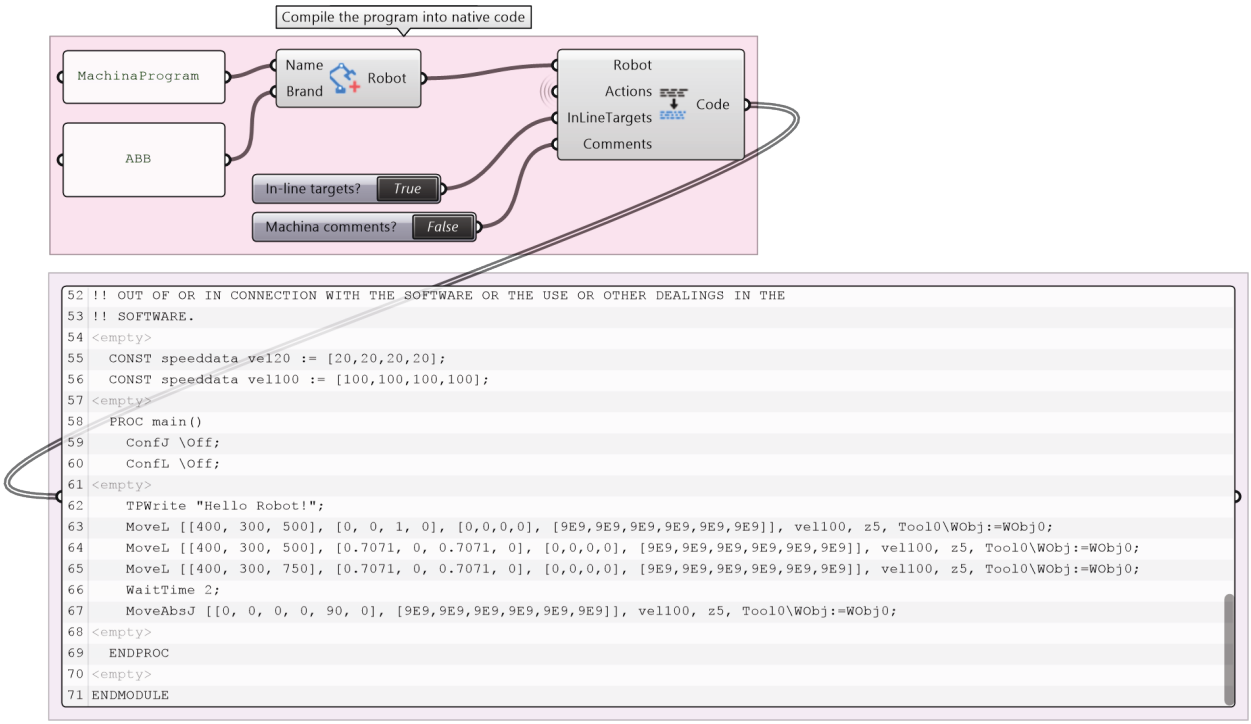
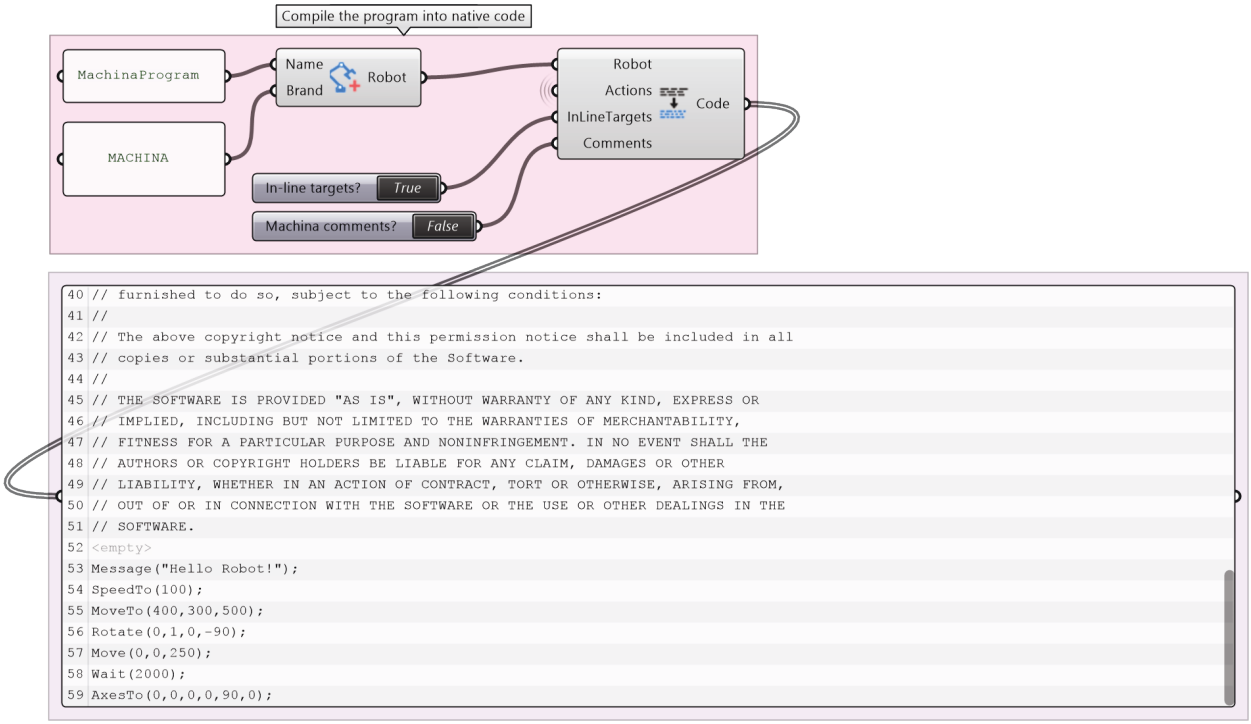


Figure 36 - Zoom into the program composition node in "Hello Robot"; actions are converted by default to human-readable string representations.




```
52 !! OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
53 !! SOFTWARE.
54 <empty>
55   CONST speeddata vel20 := [20,20,20,20];
56   CONST speeddata vel100 := [100,100,100,100];
57 <empty>
58   PROC main()
59     ConfJ \Off;
60     ConfL \Off;
61 <empty>
62     TPWrite "Hello Robot!";
63     MoveL [[400, 300, 500], [0, 0, 1, 0], [0,0,0,0], [9E9,9E9,9E9,9E9,9E9,9E9]], vel100, z5, Tool0\WObj:=WObj0;
64     MoveL [[400, 300, 500], [0.7071, 0, 0.7071, 0], [0,0,0,0], [9E9,9E9,9E9,9E9,9E9,9E9]], vel100, z5, Tool0\WObj:=WObj0;
65     MoveL [[400, 300, 750], [0.7071, 0, 0.7071, 0], [0,0,0,0], [9E9,9E9,9E9,9E9,9E9,9E9]], vel100, z5, Tool0\WObj:=WObj0;
66     WaitTime 2;
67     MoveAbsJ [[0, 0, 0, 0, 90, 0], [9E9,9E9,9E9,9E9,9E9,9E9]], vel100, z5, Tool0\WObj:=WObj0;
68 <empty>
69   ENDPROC
70 <empty>
71 ENDMODULE
```

Figure 37 - The "Hello Robot" program compiled to ABB RAPID language.




```
40 // furnished to do so, subject to the following conditions:
41 //
42 // The above copyright notice and this permission notice shall be included in all
43 // copies or substantial portions of the Software.
44 //
45 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
46 // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
47 // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
48 // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
49 // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
50 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
51 // SOFTWARE.
52 <empty>
53 Message("Hello Robot!");
54 SpeedTo(100);
55 MoveTo(400,300,500);
56 Rotate(0,1,0,-90);
57 Move(0,0,250);
58 Wait(2000);
59 AxesTo(0,0,0,0,90,0);
```

Figure 38 - The "Hello Robot" program compiled to Machina Common Language instructions.

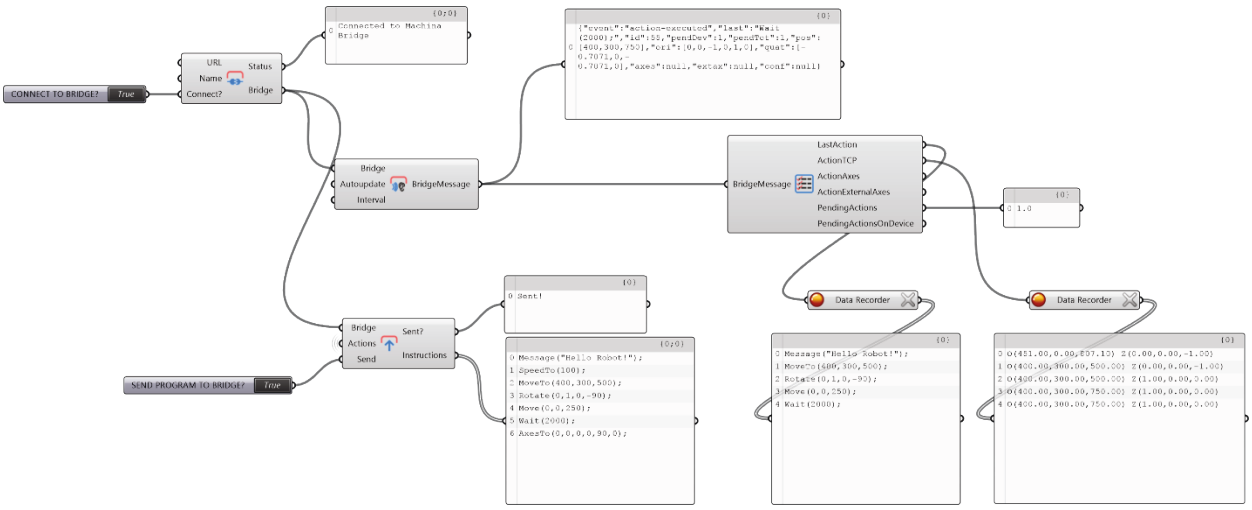


Figure 39 - Connection to the Machina Bridge from the Grasshopper3D environment: once a successful connection is established with the Bridge, actions can be sent in batches to the Bridge with the `Send` component, while the `Listen` component receives event updates from the Bridge.

In line with the spirit behind the rest of the projects in the Machina ecosystem, the VPL implementations surrogate the state representations and low-level machine control to the Bridge, only interfacing with it via actions and events (see §4.2.2). The `Bridge` category provides access to VPL-Bridge communication functionality: The `Connect` node allows client configuration for the WebSocket connection, returning an instance of a "Bridge Connection" object. This instance can be used by the `Send` node to stream a batch of actions to the Bridge via their string instruction representation. The `Listen` node also uses the connection object to listen to events coming from the Bridge at specified intervals[22]. Further nodes such as `ActionExecuted` or `MotionUpdate` parse the incoming JSON objects from the listener, and convert them into native geometry objects in the host environment (Figure 39). The responsiveness of this system is limited by the capacity of the host to perform graph updates at the interval rate.

The architecture of this hybrid concurrent control system has a very interesting property related to the enactive model. DynamoBIM and Grasshopper3D are visual programming environments where there is no need to precompile any program before execution; any change made to the relational tree of components triggers an automatic succession of downstream updates, with their effect immediately applied to the geometry, viewport or whichever other output the program is meant to affect. This creates a rather flexible programming environment with immediate feedback. This environment, combined with the asynchronous nature of the WebSocket communication with the Bridge, allows a multi-modal programming experience where a full routine can be implemented modularly, sent in small batches, and changes to the program can be made on the fly as the program executes. Programs built in this manner can range anywhere from a completely automated procedure, to a unidirectional control panel composed of buttons, to a hybrid between direct control and

[22] DynamoBIM handles these cycles with global periodic update settings, while Grasshopper components incorporate this functionality embedded.

automated responses, with a human on the decision-making loop —perhaps a sort of new "programming-on-the-loop" paradigm. Examples of this interaction model will be further discussed in §0.

4.3 Summary

In this chapter, two sample technical implementations of the *Enactive Robotics* model have been presented, developed specifically for systems composed of six-axis industrial robotic arms.

The first project is called *Machina.NET*, and is a self-contained, full-architecture implementation of the action-state model described in chapter 0, developed as a C# library. In this project, a full API of actions was designed with simple English verbs that mainly denote requests for spatial motion, as well as a collection of events notifying the program of meaningful changes in program execution. Additionally, a system state representation was developed, including a layered structure of robot execution states called cursors. Details about low-level communication with the device and robot control subrogation were provided. The goal of this implementation is to prove the power of the model to enact concurrent control of mechanical devices, seamlessly and universally; its target audience is technically-savvy individuals, familiar with computer programming and application building, but novice to the world of robotics.

The second implementation is an ecosystem of modular interfaces part of the global *Robot Ex Machina* project. The project is built around the centrality of the *Machina Bridge*, a stand-alone application built with the .NET project at the core, which exposes its functionality via a graphical user interface. The Bridge provides real-time feedback about the state of the robot, and provides a command-line interface that allows immediate actions requests to the device. Such action requests should be written as instructions in the *Machina Common Language*, a standard syntax that mirrors the .NET API, facilitating serialization and action exchange between different members of the ecosystem.

The central role of the Bridge is manifested in its potential function as mediator between machines connected to the system and other applications on the host controller. This functionality is enabled by accepting incoming client connections via the WebSocket protocol, and receiving actions requests as string messages in the Machina Common Language syntax; clients are also notified in return of execution events. Several implementations of such clients are presented in this chapter, including samples for the Processing, Python and JavaScript languages, as well as for the Grasshopper3D and DynamoBIM platforms.

Enactive systems built with these applications feature a modular architecture where the Bridge becomes the de facto state model and machine controller, and the clients act as agents interfacing with the system through the action model. The goal of this implementations is to prove the power of the enactive robotics model to be accessible, immediate, cross-platform and extensible. The target audience are creative individual with little to no technical proficiency, willing to experiment with real-time robot programming and control.

All the projects described in this chapter are open source, and can be found in the main Robot Ex Machina repository (García del Castillo 2016). These projects will serve as the base for the case studies described in chapter 0, and the controlled user study in chapter 0.

5 Case Studies

The work described in chapter 4 constitutes a sample of possible technical implementations following the *Enactive Robotics* model for the creation of robot control systems. The postulates of the model served as the guiding principles behind the design of the *Robot Ex Machina framework*, an ecosystem of libraries, plugins and applications specifically geared towards industrial robotic arms. Each component within the framework is catered to address the needs of a specific group of users, covering as a group a wide range of backgrounds, levels of technical expertise and platforms of choice. Moreover, the consistent design and integrated nature of each component makes them particularly suitable as building blocks of modular architectures, providing users with a flexible and adaptable suite of tools to build robotic systems with.

The main thesis in this dissertation is that robot control systems designed following the principles of the Enactive Robotics model feature two main advantages over traditional ones: they provide an easier entry point for novel users to the development of robotic systems, and they constitute a framework which systematically provides a richer and deeper scope of possibilities for experienced users. While some evidence of the first assertion will be hereby presented, support for this claim will be discussed mainly in the chapter 0. Instead, this section will situate the conversation around the second premise.

In this chapter, a collection of projects created with the tools described in chapter 0 will be presented. The purpose of illustrating these examples is to demonstrate the capacity of control systems built after the principles of the Enactive Robotics model, such as the Machina framework, to enable the creation of robotic systems that supersede traditional programming paradigms. These systems provide enhanced interaction possibilities, elevate the level of concurrency on the system, and provide flexible and adaptable solutions to complex problems. In this sense, the Machina framework is studied as an instrument to understand the validity of the Enactive model and its postulates.

The projects presented in this section constitute a sample of the work in interactive robotics developed by the academic and professional community during the course of this research. Some of the case studies hereby

discussed were prototypes designed by the author, some of them stemmed from workshops he co-led, and some were developed independently by other groups of designers[23]. This collection covers a wide variety of author backgrounds, robotic applications and interaction paradigms, and helps illustrate the versatility of the model to adapt to a wide variety of system architectures and application requirements. The case studies are presented descriptively, characterizing the architecture of each system, the relation between their parts, and the use case scenario they represent. The projects have been loosely organized in relation to the nature of their outcomes, and their evolution over different iterations. The section concludes with a discussion around the characteristics these projects exhibit in relation to the properties of Enactive Robotic systems, and a case is made for their validity to support the main thesis of this dissertation.

5.1 Projects

5.1.1 Tracing

Figure 40 - The "Drawing with Robots" installation.

[23] For a full breakdown of project authors, credentials and timelines, please see "Appendix A: Project Credits."

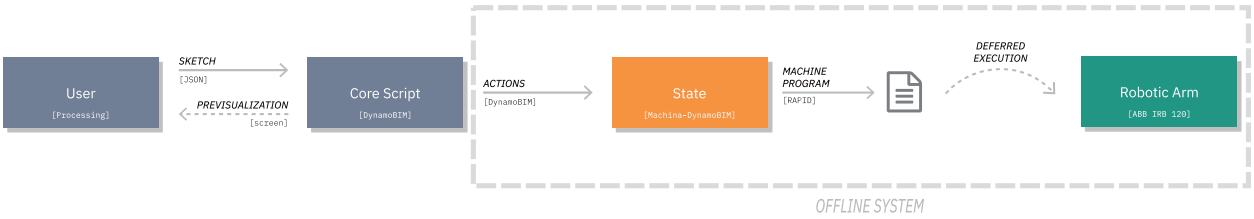


Figure 41 - System architecture of "Drawing with Robots."

One of the earliest public projects developed with the Robot Ex Machina framework was the ***Drawing with Robots*** installation for Kids Day at Autodesk 2017 (Figure 40). In this work, the authors[24] designed a robotic drawing system to allow visiting children and teenagers to use a small industrial robotic arm, with a marker attached to it, to draw doodles on paper from a screen interface.

At the time of this project, the Machina.NET library was quite advanced, as well as offline-only versions of the VPL implementations. However, the Bridge application had not been developed yet, so an ad-hoc solution became necessary to connect an external drawing application to a C#-enabled environment, capable of providing the robot compilation capacities of the project. Time constraints prevented the authors from building a pure .NET application, needing to opt for frameworks that already incorporated interactive graphic rendering and advanced geometry processing tools.

The chosen solution consisted of a modular architecture of applications, outputting a robot program file to be executed offline on the device (Figure 41). The system consisted of a touchscreen-enabled computer running a bespoke drawing application built in Processing, providing participants with an UI for simple stroke plotting. The interface also provided simple digital drawing tools, like "undo" and "clear all". As participants decided the drawing was done, a "send to robot" button gave users the possibility of signaling the desire to plot the drawing on the robot. However, this button saved the strokes as point arrays in JSON format to a file to be used as data exchange between Processing and a DynamoBIM script. The script served the following main functions: transform stroke coordinates from pixel coordinates in screen space to robot coordinates in paper space, add travel points plus pen up and down operations, pre-visualize the robot motion and compile the program into a file in the RAPID language for an ABB IRB 120 robot. The program file would then be manually loaded to the robot and executed by the supervising authors.

The project resulted in a successful morning of children engaging with robots and fostering STEM culture. Additionally, the offline program execution turned convenient given the extreme safety measures required for this audience. However, the project manifested the early need for a systematic solution for cross-platform connectivity to the robot; the authors were able to build all the individual pieces and connect them properly, but the solution became technically intense, and was not generalizable to other possible scenarios.

[24] Unless otherwise stated, the term "author/s" will be used in this chapter to refer to the creator/s and other contributors of the projects described in the section, instead of the customary reference to the author of this dissertation.

Furthermore, the disconnect between stroke design and execution prevented providing further feedback to the users, such as drawing progress, or on-the-fly changes to the drawing.

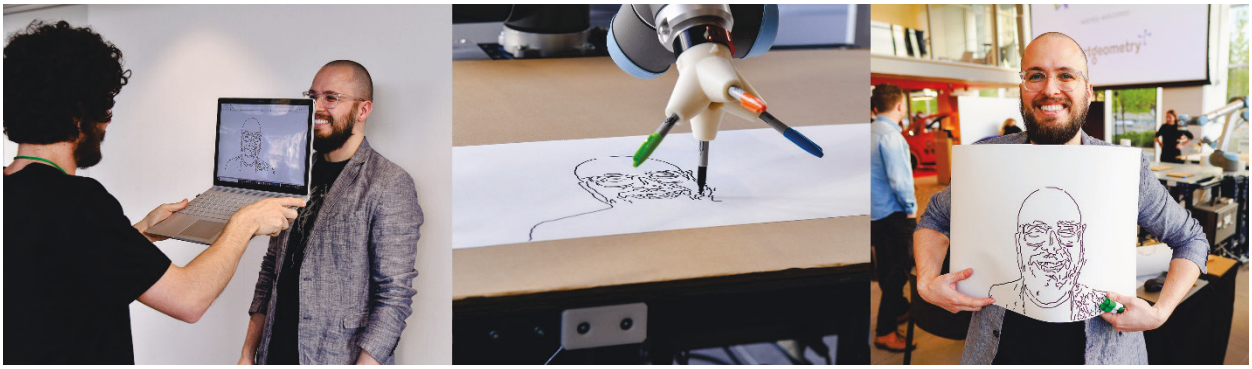


Figure 42 - The "Selfie Plot" installation.

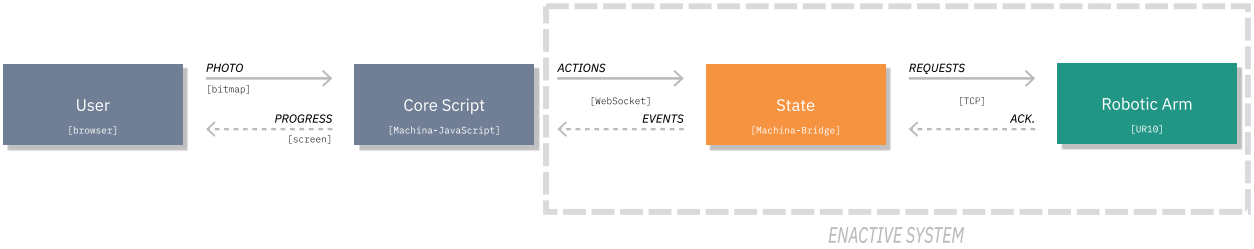


Figure 43 - System architecture of "Selfie Plot."

The idea of interactive robot plotting became further refined on a second iteration of the concept, with the ***Selfie Plot*** application at the Mind Ex Machina workshop in the SmartGeometry Conference 2018 (Figure 42). In this project, the author designed an installation to allow conference visitors to take a self-portrait photo with a stage camera, and have a robot plot a stroked version of the image with a maker.

An initial version of the Bridge had been developed for testing at the workshop, in order to address the anticipated needs for educational and cross-platform compatibility tools (see Machina Bridge). As a result, the author was able to concentrate most of his effort on creating the logic and front-end implementation of the installation. His solution became a web-based application that could use a device's integrated webcam to take a photo of the user on demand. A canny edge detection algorithm would give the user a real-time previsualization of the stroked version of the portrait, and prompt them to choose their preferred snapshot. Upon verification, the application would perform a similar robot-preparation process as in "Drawing with Robots:" transform the strokes from screen to paper space, and add travel plus pen up/down operations. However, in this project, the system was connected online to a Universal Robot UR10 via the Bridge application, and strokes were sent concurrently to the machine in batches, by streaming Machina Common Language (MCL) action strings to the Bridge using the Machina for JavaScript library. Furthermore, the web application received event notifications from the Bridge and used them to track plotting progress and highlight it visually on the app (Figure 43).

The final installation granted a rather interactive experience for users, as the feedback provided by the dynamic photo-to-strokes conversion gave users a playful experience, as well as a fairly accurate representation of the final physical outcome. The system also fostered fluid communication between the application and the robot, overcoming the need for manual program uploading —a human was still on the loop just for safety supervision—, and adding the possibility of real-time plotting of drawing progress application-side. Finally, the project showcased the potential of the framework to create self-contained robotic applications that could be deployed globally on the web, but used to interface with a local machine through mediation of the Bridge.

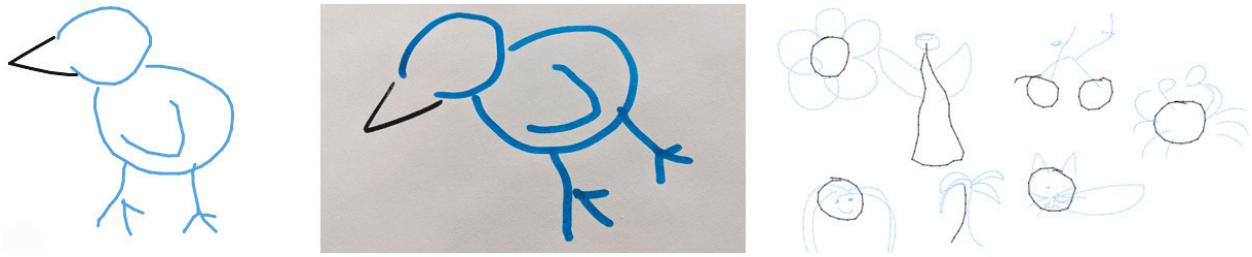


Figure 44 - Sketches from the "RobotSketch-RNN" installation. A black stroke can be input to the system, and neural network will generate a suggestion in blue (left). Upon user approval, the robot will draw the sketch on paper (center). Different prediction models can be selected on the interface, such as flower, angel, bicycle, crab, palm tree or cat (right).

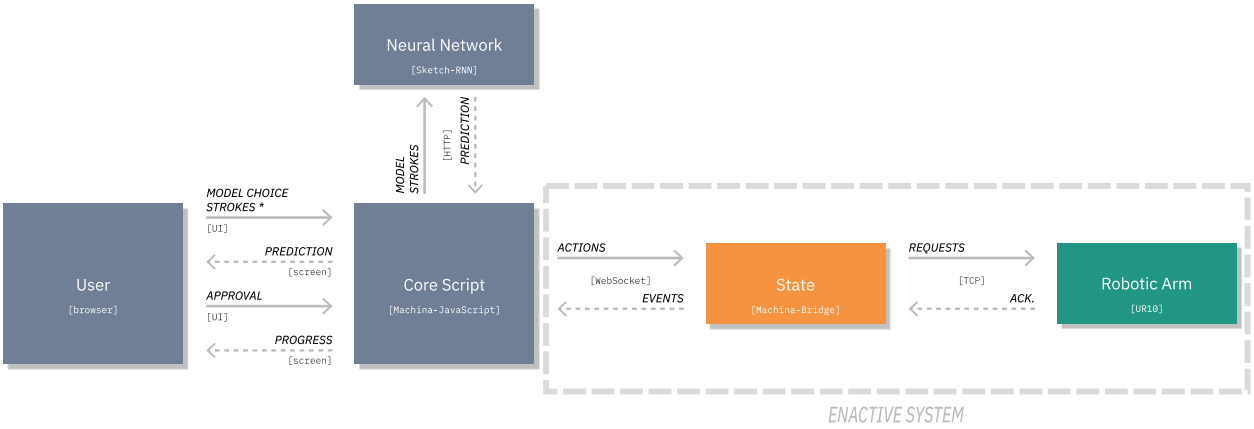


Figure 45 - System architecture of "RobotSketch-RNN" and (*) "Exquisite Corpse." In the latter, stroke input was only possible upon the first sketch.

"Drawing with Robots" and "Selfie Plot" were both founded on the notion of robot drawing from direct human input. However, the modular approach fostered by the Robot Ex Machina framework provides greater flexibility, with the possibility to accommodate further decision-making agents in robotic systems. Such idea was explored in two further robot-drawing installations, featuring collaboration between humans and artificial intelligence (AI) systems.

RobotSketch-RNN was an installation developed in preparation for the abovementioned Mind Ex Machina workshop (Figure 44). The project was inspired by the work of one of the authors in human-AI collaboration (Martínez 2017), and enabled by the publication of the `Sketch-RNN` framework, "a recurrent neural network (RNN) able to construct stroke-based drawings of common objects" (Ha and Eck 2017).

The project consisted of a robot drawing application, plotting sketches initiated by user input and further completed from suggestions coming from the neural network. The application used a stroke-drawing interface similar to "Drawing with Robots," ported to a web-based application. The application was also connected to a custom HTTP server developed by the authors[25], serving prediction requests based on the `Sketch-RNN` neural network, and using the project's database of trained models. This framework was extremely convenient for the project, as its vector-based prediction model was a great fit for the linear nature of robotic motion[26]. The main application was using Machina for JavaScript to interface with an ABB IRB 1200 robot via the Bridge application (Figure 45). The robot had a custom holder tool with capacity for four markers.

The interface prompted the user to draw the first stroke of a pre-determined figure from the model library such as, for example, a bird, a bicycle, a car, a flower, etc., with the choice of target model customizable through the interface. Upon completion, the user could click "complete," in order to generate a drawing prediction from the neural network. The application would then send the stroke parameters to the HTTP server, and receive a suggestion from the neural network with further strokes on how to complete the drawing. The user would then enter a "curation" cycle, where prediction strokes would be displayed on the application UI for approval. If the user was unsatisfied with the prediction, they would have the possibility of iteratively request more predictions, change the target prediction model or change the source stroke, until a satisfactory output was reached. At that point, the user could click "draw," and the application would follow a process similar to "Selfie Plot" to generate stroke-based actions for robot drawing. The drawing would then be plotted on physical paper with the robot, using markers in different colors for the input from the human user and the prediction generated by the neural network. Additionally, drawing progress was highlighted on the application based on execution progress coming from the Bridge events.

[25] In this project, the HTTP server was running locally on the same host computer as the main application. However, the architecture of the system was designed to potentially allow this module to be hosted on a local/global network.

[26] Most image generation frameworks based on machine learning, such as Pix2Pix (Isola et al. 2016), are usually based on raster image processing, rather than vector information.

Figure 46 - Art generated by the "Exquisite Corpse" piece.

The "RobotSketch-RNN" concept was further extended on the ***Exquisite Corpse*** project during the Mind Ex Machina workshop (Figure 46). The project was inspired by the eponymous literary figure, and uses system recursion to generate a collage of sketches from the source human input.

The architecture of this project was very similar to the one described in "RobotSketch-RNN." However, the main application was here written in Processing, due to personal preference by the author. The robot was a UR10 in this case, with the same custom multi-marker holder (Figure 45). Users could choose a prediction model, draw a stroke and curate the most satisfactory sketch to be sent to be plotted by the robot. The main contribution of this project was the capacity of the user to continue subsequent iteration cycles, in order to recursively draw further sketches based on the previous one. Upon verification and plotting of the first sketch, the user would be prompted to draw a second one, using the last stroke of the previous sketch as the input for the new prediction. Cycles of human curation could be performed concurrently to physical plotting, and additional "approved" sketches could be sent to the Bridge, becoming buffered for drawing and inputs for the next one. The application would cycle over different colored markers for each sketch. In this sense, the installation became a collaborative collage, where a human could start and curate the drawing predictions of a machine learning model, while the robot was concurrently drawing them.

"RobotSketch-RNN" and "Exquisite Corpse" conform two examples of human-AI collaboration with robotic output. The former was designed to foster the creative aspect of interactive drawing and the playfulness of exploration and discovery in the interaction with an AI. The latter became a more poetic interpretation of the same principle, geared towards highlighting the complexity embedded in the training models through curated recursion. Nonetheless, both projects push the notion of augmenting the creative capacity of a human through collaboration and curation of an AI agent in charge of task automation and suggestion (Martínez 2017). Furthermore, the concurrency of the interaction with the neural network and the robotic arm contributes to the perception of agency in the system, reinforcing the ideas behind the enactive robotics model.

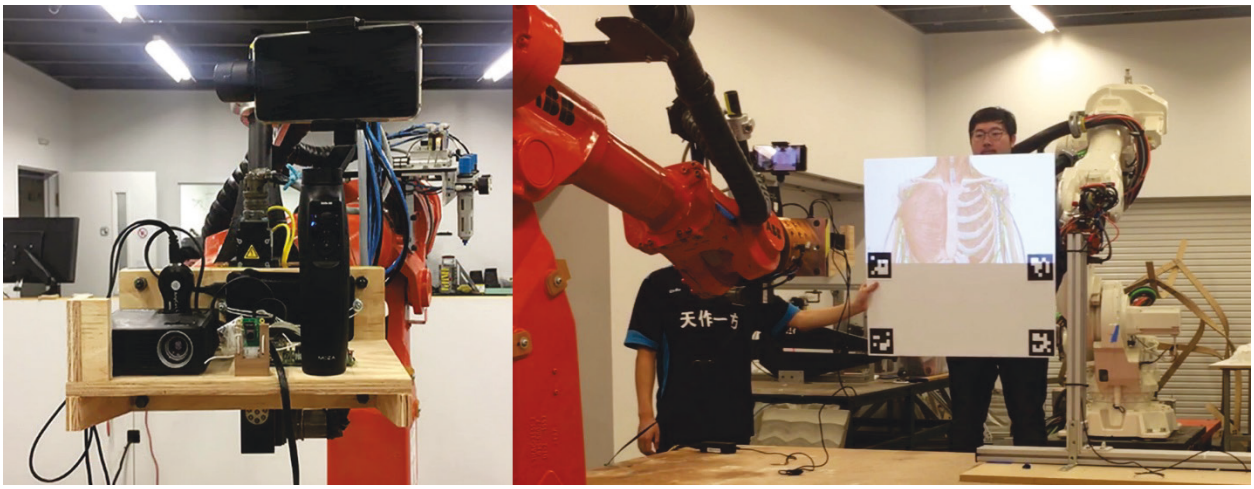


Figure 47 - The "Real-Time Anatomy" installation. A custom tool with video camera, projector and augmented reality system is mounted to the robot (left) and used to overlay anatomical images of the body on a tracked white board.

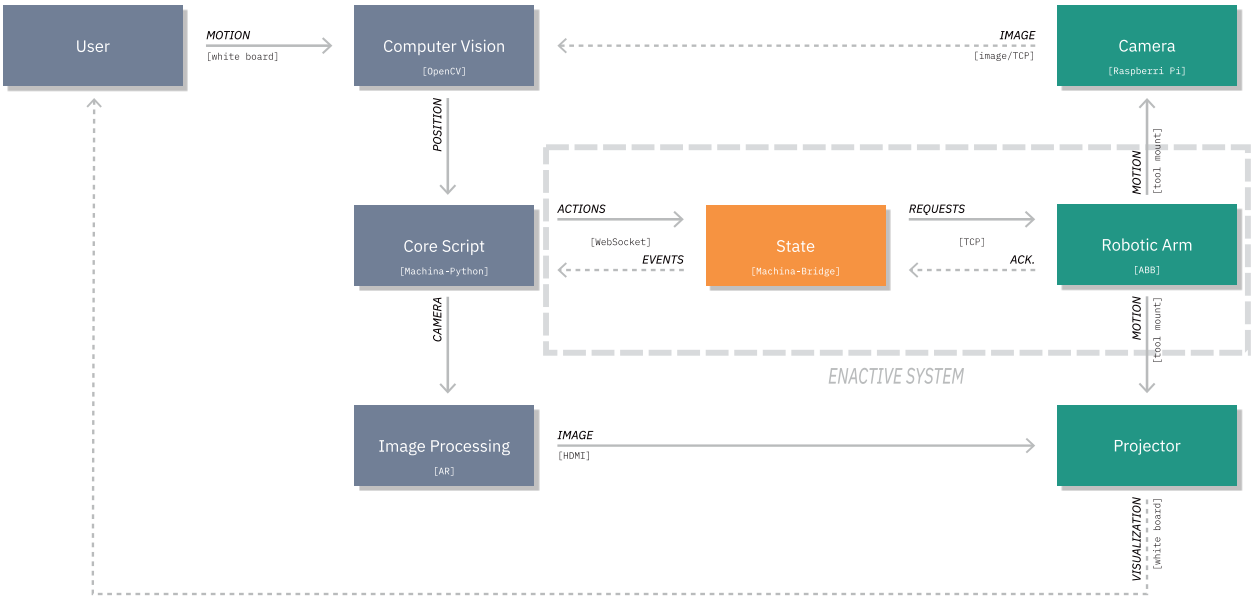


Figure 48 - System architecture of "Real-Time Anatomy."

Human-robot collaboration was further explored in the ***Real-Time Anatomy*** project by students at Carnegie Mellon University (Figure 47). In this project, the authors created an interactive robotic projection system, in order to augment the human body with an overlay of anatomical information.

The installation featured an industrial robotic arm holding a custom-made tool, composed of a calibrated video projector and a video camera (Figure 48). At the center of the installation, a human subject is standing in front of the robot, with another user holding and moving a white board around them, used as projection screen. The authors implemented a closed-loop system in which computer vision (CV) was used to detect the position of four fiducial markers at the corners of the board. The markers were used to track the spatial movement of the board by the user and reorient the robot in real time, in order to maintain constant framing of the projection over the board. Furthermore, the recomputed position of the robot was additionally translated to the motion of a virtual camera in a real-time 3D rendering environment connected to the system. The rendering generated an x-ray anatomical figure of a human body, matching the orientation of the robot projection to the screen.

This design resulted in an interactive installation where the robot could follow the motion of the board, maintain matching projection over it, and adapt the content to the virtual point of view. This enabled a controlling user to hover the screen around the body of a real human, and project over it a virtual representation of their internal anatomy. The project maximizes the capacity of the Machina framework to maintain concurrent control of the robot, being able to respond to the CV system in real-time and generate visuals according to its state —a system architecture hardly realizable under the offline model.

These projects explored different interaction models for the purpose of creating drawings on paper or projecting over a moving surface. In all of them, the role of the robot arm was as actuator for the final output, with its nature or state having little relevance to the design of the system; each installation was physically set up and calibrated to accommodate to the robot's workspace, full range of motion and kinematic reachability was assumed, and no further checks were needed.

Figure 49 - Project "Dereliction:" 3D scanning of the piece (left), pattern propagation algorithm (center) and milled boulder (right).

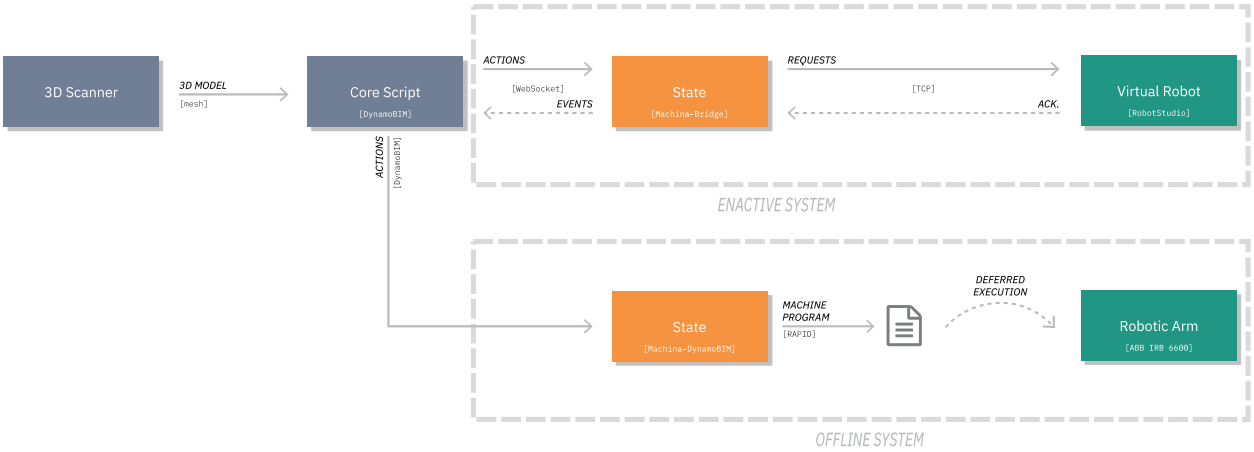


Figure 50 - System architecture of "Dereliction."

The ***Dereliction*** project for the Digital Stone Project 2018 (Figure 49) explored the possibility of inverting this hierarchy, using the robot's physicality as the core driver behind the final design. In this project, a generative design workflow was created to streamline the milling of a pattern on the surface of a 3D-scanned marble boulder.

"Dereliction" was a research project part of a marble sculpting workshop, where artists were invited to explore creative opportunities using robotic milling. Their typical workflows usually started with digital sculpting on conventional 3D modeling software, with machine instructions generated by local engineers with dedicated production software like Autodesk PowerMill. Milling typically happened on marble blocks pre-cut at the quarry to tight bounding boxes. The author proposed to recycle scrap rocks from the quarrying process instead, and use generative design to mill form on this rock, turning it again into valuable material.

For the project, a generative script was developed in DynamoBIM to import 3D scans of scrap boulders, and generate a maze pattern on its surface. The script incorporated mesh optimization operations with Mesh Toolkit, such as cleaning, reduction and remeshing. A custom graph search algorithm was developed to propagate a maze pattern over the triangles of the mesh. The pattern was discretized into toolpaths corresponding to the branches in the maze, and further mill in/out and travel trajectories between waypoints where added. The geometry of each full toolpath was then translated into a sequence of actions using Machina for Dynamo.

The process of generating complex robotic toolpaths is a fairly common operation in these kinds of VPL environments, especially when the target outcome is a program to be executed offline on the robot. However, the challenge in this project became the relation between the boulder size and the robot workspace —in this case, a large ABB IRB 6600. The rock was fairly large in comparison with the arm —roughly 1000x550x400 mm—, and was scanned already on its final position for milling. This situation made robot reach a critical issue, as a significant portion of the milling targets were close to the arm's reach limit, or even past it. As most of the milling was scheduled to happen overnight, an unsupervised out-of-reach error in the middle of the process would mean the wasting of several hours of robot time until the morning.

In anticipation to possible errors during fabrication, the real-time capacity of the Machina framework was used to perform a simulation prior to milling, identify problematic toolpaths and prune them from the final program (Figure 50). The generative script was connected via the Bridge to a RobotStudio simulation of the robot and the scanned boulder on its milling position. Then heuristically, toolpaths were streamed to the simulation one at a time. If the toolpath was successfully completed by the simulation, it would be flagged as valid on the script; if the simulation yielded an out-of-reach error instead, the toolpath would be trimmed from the final pattern. The final clean milling program was entirely verified again through simulation, exported to a RAPID file from Machina for Dynamo, and executed offline on the robot[27].

The final outcome of the project was the scrap boulder turned into a sculpture, with a network of maze lines milled on the surface, concentrating on the area where the robot had dexterity of motion [FIGURE ##]. But most importantly, the project showcases the capacity of the enactive robotics model to include virtual robots as valuable parts of holistic robotic systems, using concurrency to incorporate them in simulation, verification and safety checks workflows.

[27] The ABB robot arm did not feature the "616-1 PC Interface" option, necessary to perform TCP socket communication with the Machina framework.

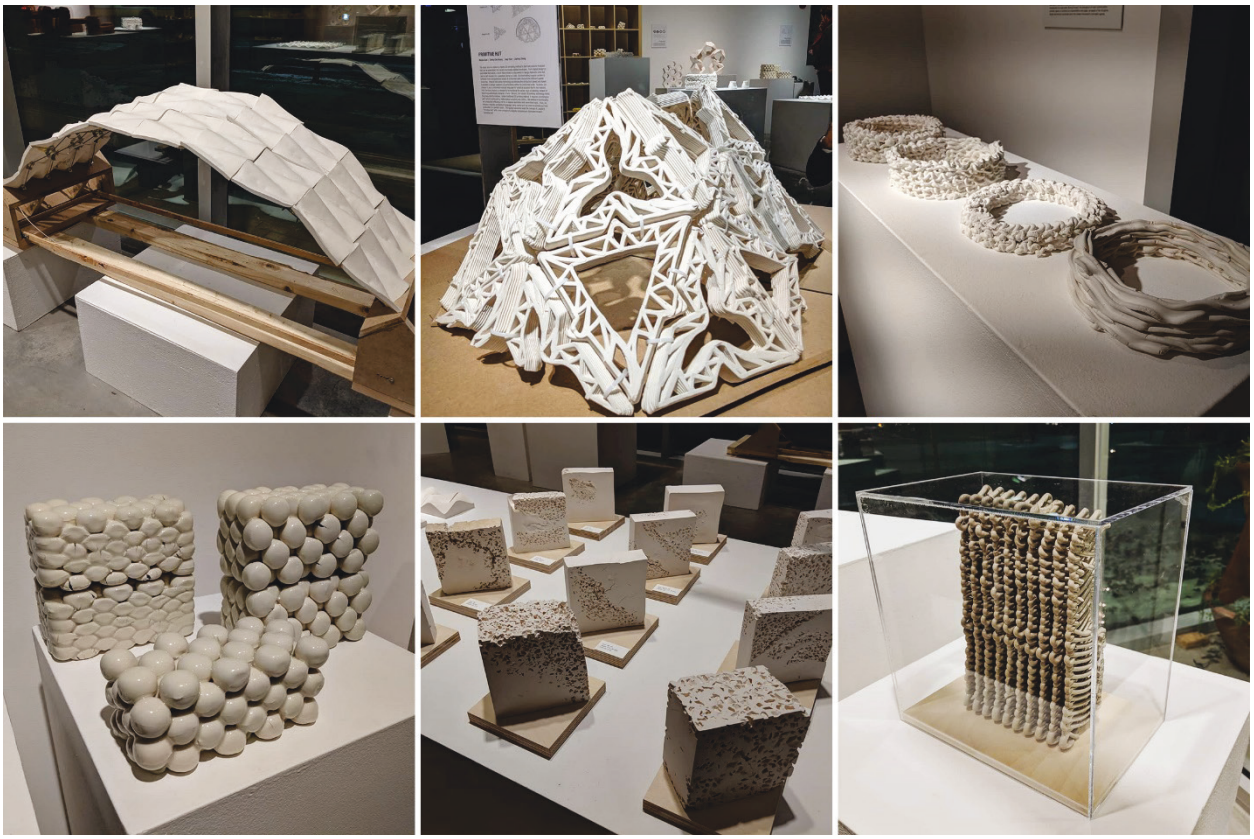


Figure 51 - Robotic fabrication projects from the "Material Systems" class. From top to bottom, and left to right: "Ceramic Arch," "Geodesic Dome," "Pappardelle Pillars," "Soft Aggregate Jamming," "The Mixology Table" and "Spatial Print Trajectory."

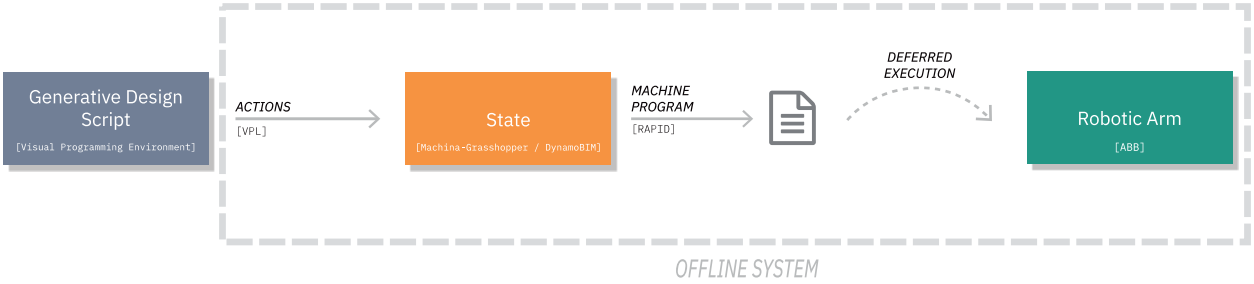


Figure 52 - General architecture of the "Material Systems" projects.

5.1.2 Fabricating

The capacity of the Robot Ex Machina framework to be used for fabrication projects was also put to the test in the early stages of this work. The use of industrial robotic arms as advanced, customizable CNC machines for product manufacturing at various scales became increasingly popular in the last decade, with schools around the world —especially architecture ones— setting up dedicated robotic fabrication facilities (Gramazio and Kohler 2014). A bloom of software tools and scripting packages became enablers of robotic fabrication projects, published in new dedicated conferences on the topic (see §2.3). The proficient use of industrial robotic arms as powerful and versatile fabrication tools by an increasingly large community of designers constituted the perfect benchmark to evaluate the capacity of the Machina framework, both in providing a solid foundation for conventional fabrication workflows, while at the same time, expanding the scope of creative possibilities due to its concurrent nature.

The context for the first large test run was the ***Material Systems: Digital Design and Fabrication*** class, in the Fall of 2017 at the Harvard Graduate School of Design (Figure 51). The course would "position ceramic material systems as a vehicle for exploring applied research methodologies and investigation into the opportunities (and challenges) afforded by digital fabrication techniques," advancing "strategies for robotics, additive manufacturing, and other computational fabrication technologies" (Bechthold and García del Castillo 2018).

The course featured a range of robotic fabrication projects, focusing on ceramics as the primary material medium. In ***Ceramic Arch***, the authors used a robotically actuated pin mold to create temporary formwork for tile pressing, in order to build a post-stressed arch out of thin curved ceramic tiles. Additive manufacturing with clay was the method used in ***Geodesic Dome*** to fabricate custom interlocking blocks, to be assembled into a geodesic dome. The authors of ***Pappardelle Pillars*** also used layered deposition as a medium to explore the effect of variable nozzle shapes, extrusion feed rates and robot speeds in the formal qualities of 3D-printed circular columns. In ***Soft Aggregate Jamming***, the robot was used for the controlled pressing of spatial aggregates of fresh hollow clay spheres, resulting in stackable bricks of variable density and thermal properties. Finally, in ***The Mixology Table***, the robot was used for the controlled continuous rotation of a plaster mold filled with a custom mix of clay slip and hydrogel. By experimenting with rotation angles, degrees of freedom and time/speed intervals, the authors were able to design gradients in the distribution of the suspended hydrogel, resulting in fired elements with variable porosity.

At the time of this course, the core Machina.NET library was available with full concurrent capabilities, as well as the DynamoBIM and Grasshopper packages in offline-only form. However, despite the dedicated training and office hours, all the above-mentioned projects were implemented in conventional offline mode, developed within the authors' choice of VPL (Figure 52). Multiple reasons can be attributed to this preference: the lack of proficiency in computer programming, the novelty of the robot as a fabrication tool, the material emphasis of the course, etc. Most of the students inquired about this choice manifested affinity for the use of VPLs as a design environment, and expressed the desire to capitalize on a digital tool they were proficient at,

concentrating their learning efforts on the physical aspects of their projects. Nevertheless, it is noteworthy that in two of these projects, the relative flavors of the Machina actions (see Action Model) were extensively employed —a feature not present in similar VPL robot tools. The authors manifested how this feature had enabled them to easily express the logics of their programs in relation to the configuration of the robot ("The Mixology Table"), or fluidly explore formal variations in the design of their toolpaths in a way that would have been seemingly tedious to compute in absolute terms ("Pappardelle Pillars").

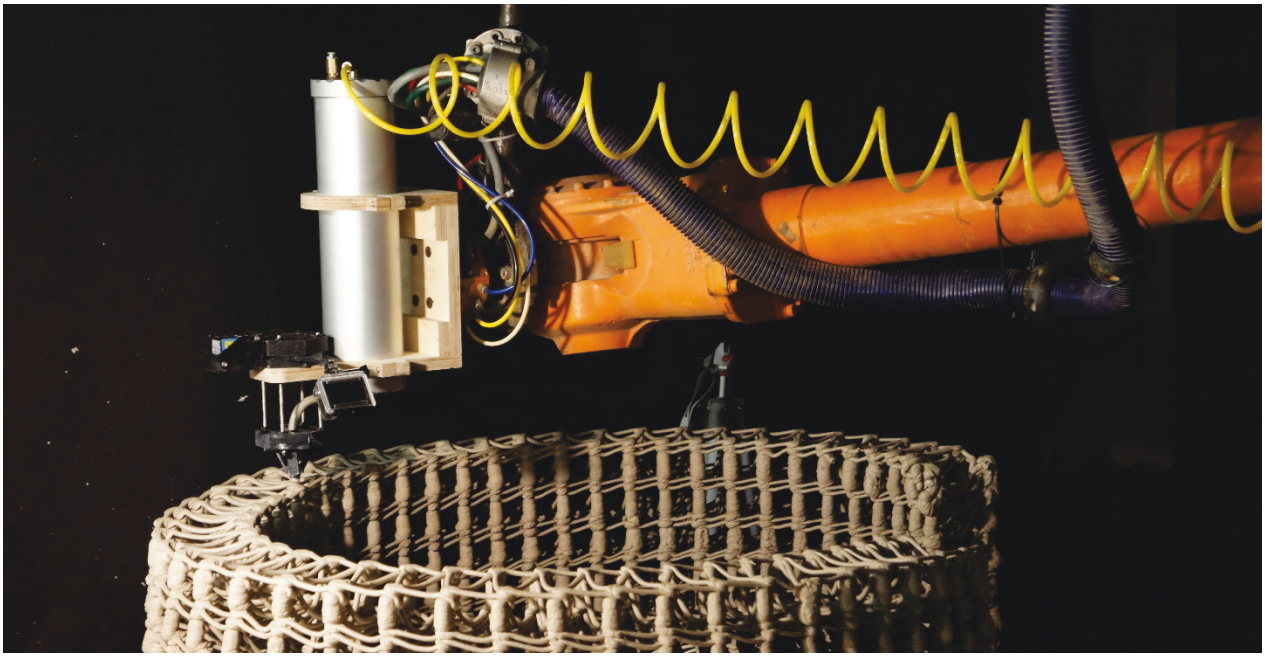


Figure 53 - Snapshot of the final prototype on "Spatial Print Trajectory" series.

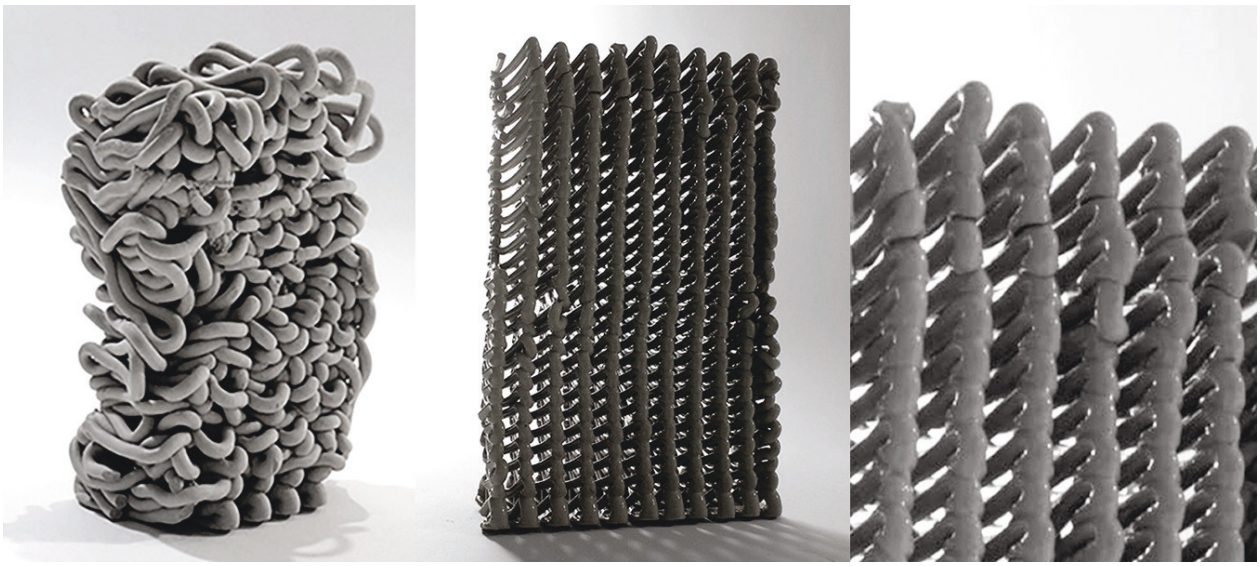


Figure 54 - Early prototypes from the "Spatial Print Trajectory" project. The cumulative deformation and shrinkage of the printed layers often resulted in uncontrollable, failed prints (left). The authors were able to compensate for the unpredictability of the system with a combination of geometric redundancy and manual readjustment of the 3D print model based on deflection measures (center and right).

The "Material Systems" course was also the starting point for the work conducted as part of the ***Spatial Print Trajectory*** project (Figure 53). The aim of this research was to create a procedural system for the generation of robotically 3D printed spatial lattices out of clay.

During the first phase of this work, the authors conducted a comprehensive exploration of different permutations of spatial toolpath configurations, material properties and extrusion speeds (AlOthman et al. 2018). The printing procedure was composed of three steps: a layer of vertical curls formed with a custom drag-and-anchor technique, a horizontal layer to stabilize the previous one, and a third pass with a heating gun mounted on the extruder to harden the material.

However, during the printing process, the nature of the material mix and its possible irregularities would cause local deviations on the shape of the deposited bead. Furthermore, the accumulation of layers of fresh material would cause it to sag. The cumulative effect of these deformations would grow exponentially over the layers, resulting in uncontrollable deflections from the original model, and often failed prints (Figure 54). The authors were able to compensate for the unpredictability of the system with a combination of geometric redundancy and manual readjustment of the 3D print model based on deflection measures, generating cycles of offline code per printed layer with Machina for Grasshopper (Figure 52). Nevertheless, this resulted in a rather laborious and slow process that involved significant manual labor.

In the second iteration of the project, the authors focused on developing a closed-loop 3D printing system, with integrated material sensing and dynamic self-recalibration. For this purpose, a fourth step was added to the original printing procedure, where the deflection of the anchor points of each layer was measured with a laser distance sensor mounted on the extruder tool. The computed difference between theoretical and

deformed positions would be fed back into the digital model, and a new custom toolpath would be generated to compensate for local irregularities (Figure 55). Such addition would require a robotic system that could be monitored concurrently, in order to be able to match robot positions and sensor readings. Additionally, there was a desire from the authors to fully automate the printing procedure, in order to speed up the process and minimize tedious human intervention.

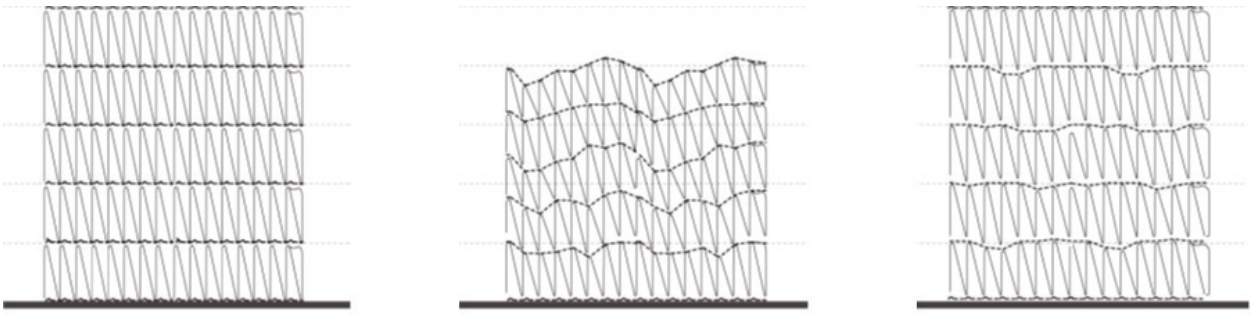


Figure 55 - Dynamic recalibration of the "Spatial Print Trajectory." The digital model generates perfectly geometrical toolpaths for ceramic 3D printing (left) which deform and sag during the printing process (center). The "Responsive Spatial Print" project incorporates real-time sensing of material deformation, and generates toolpaths to compensate them (right).

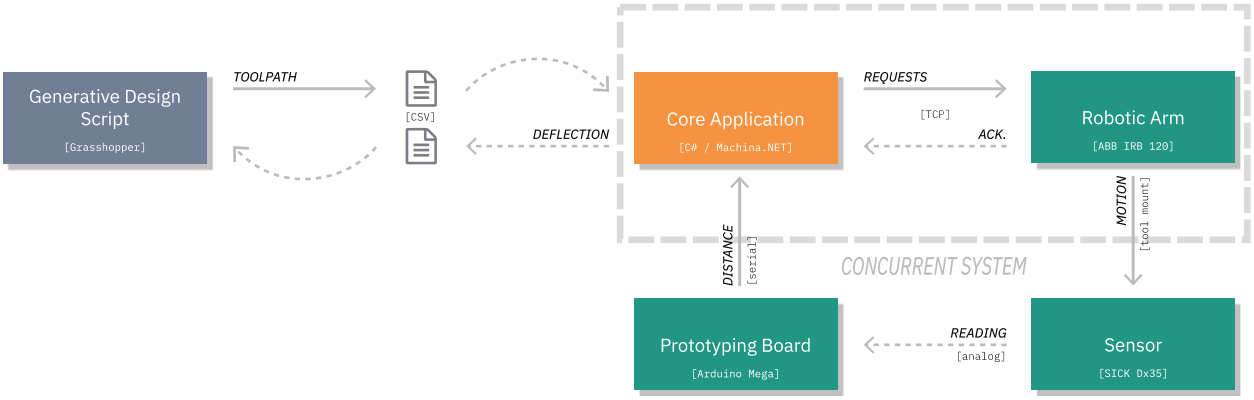


Figure 56 - System architecture of "Spatial Print Trajectory."

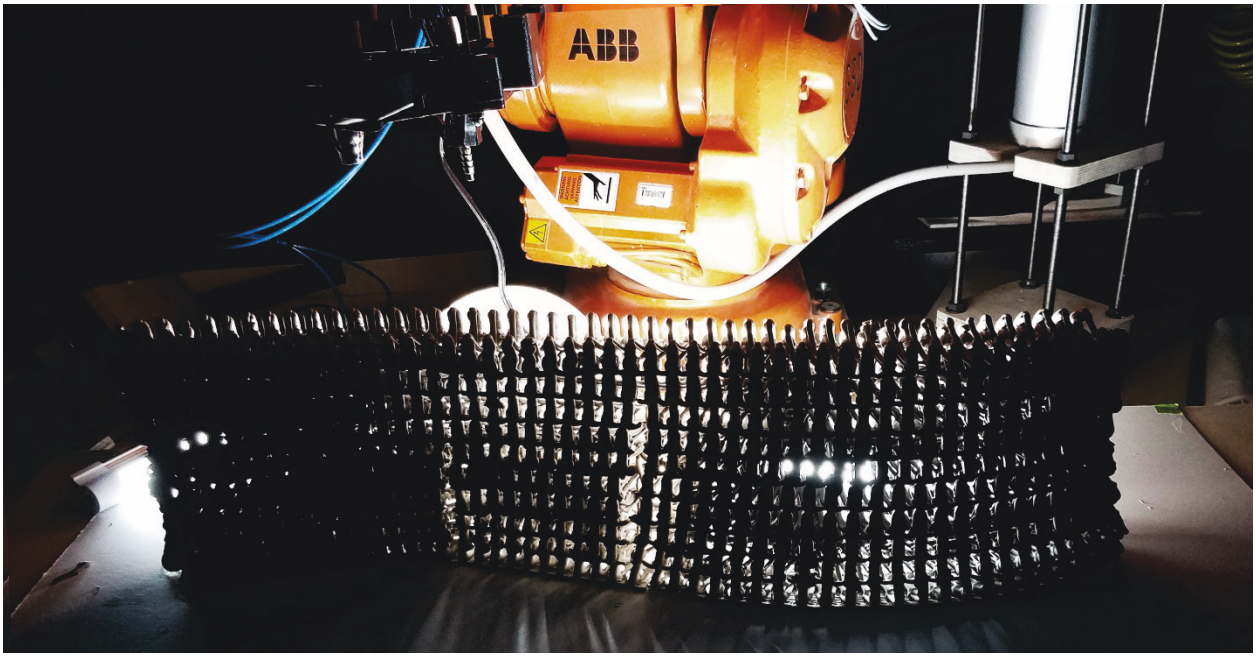


Figure 57 - First recalibrated prototype of the "Responsive Spatial Print" project.

At that time, Machina for VPLs worked in offline mode only, so an ad-hoc solution to concurrently connect the existing 3D printing logic to the robot was necessary. A modular architecture was designed around a core C# application built on top of the Machina.NET library (Figure 56). The system still relied on the computational geometry capacities of the previously developed Grasshopper scripts, and interfaced with them via periodic exchange of geometric data. The 3D printing workflow was implemented as follows:

- The Grasshopper script generates the waypoints and speeds for the printing, heating and sensing toolpaths of the first layer.
- The data is written periodically to comma-separated values (CSV) text files.
- Changes in the CSV file are detected by the core application and loaded.
- The application converts toolpath data into action sequences, and streams them to the robot.
- During the sensing phase, the application is continuously listening to distance data from the sensor through serial communication with an Arduino board, using the Firmata protocol (Steiner 2009). The application synchronizes incoming data with acknowledgment messages from the robot to generate a list of per-anchor deflection distances.
- Sensor data is written to an additional CSV file.
- Changes in the sensor data file are detected by the Grasshopper script and loaded.
- The script uses the computed deflection values to generate the toolpaths for the next layer, with the printed geometry compensating for local deformations on a per-anchor basis.

The result of this implementation was ***Responsive Spatial Print*** (Im et al. 2018), an improved iteration of the ceramic 3D printed spatial lattice, incorporating dynamic recalibration based on print deformation (Figure

57). The architecture of the project constitutes an early manifestation of the principles of the enactive model, where the core C# application acts as the central state representation and mediator with the robot, and an external decision-making agent can interface with it via a common standard of CSV files.

This phase of the "Spatial Print Trajectory" research also served to identify several challenges still to be addressed. The development of the modules involved in the system required significant technical expertise, including building windows-based applications and handling low-level communication with the sensor, which made the process slow and prevented cross-team development efforts. The specificity of the core application as central translator between geometry and actions also constituted a rather rigid solution, which required changes in the 3D printing logic to be reflected in the core application, detracting from the development flexibility typical of such modular architecture. Finally, communication with the Grasshopper script in this system is only pseudo-concurrent, as the use of an intermediate data file relies on periodic updates from both parties, introducing lag in the system and becoming a solution prone to errors.

Figure 58 - Full-scale prototype of the "Responsive Spatial Print" project.

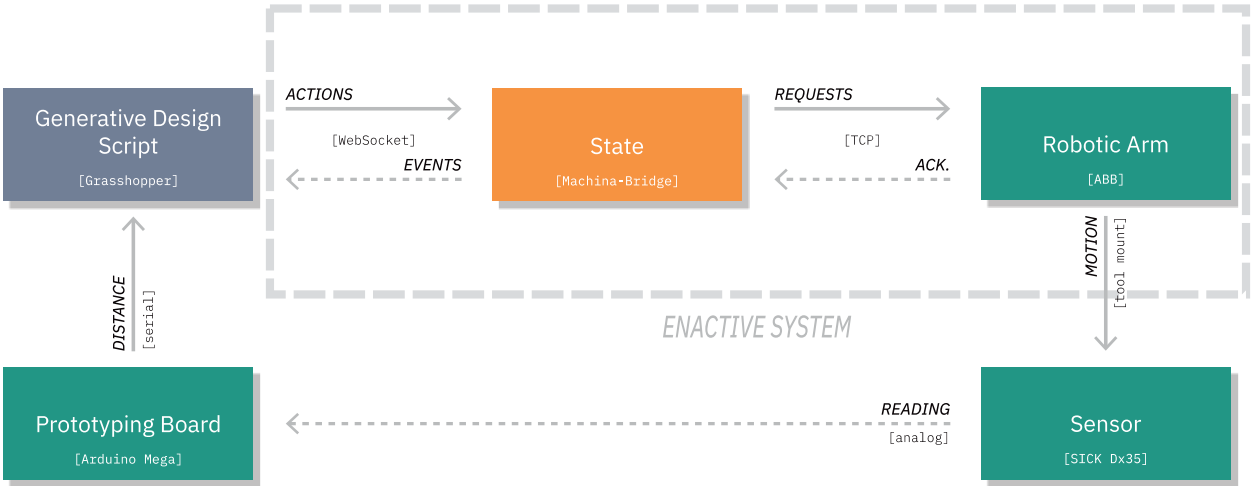


Figure 59 - System architecture of "Responsive Spatial Print."

The project highlighted the need for a more systematic solution that could overcome these problems and could be generalizable to similar scenarios. The development of the Machina Bridge (see §4.2.1Machina Bridge) enabled the possibility of expand the logic of the enactive robotics model to any external framework, including visual programming languages. As a result, Machina for Grasshopper and DynamoBIM were further developed, to feature connection methods to the Bridge, and develop a responsive spatial print solution without the need for an intermediate, project-specific robot mediator.

On the third stage of the project, the authors were able to implement the toolpath conversion into action-based programs natively in their Grasshopper script with the corresponding Machina interface, and stream them directly to the Bridge without the need for intermediate exchange files (Figure 59). Furthermore, sensor input could also be incorporated directly into the Grasshopper script via the Firefly plugin (Payne and Johnson 2013), shifting all the printing-based logics to the VPL environment and turning the robotic modules as project-agnostic. Further material studies by the authors and refined toolpath generation logics led to a significant increase in production scale, producing a sample print of roughly five feet tall and three feet in diameter (Figure 58).

Figure 60 – Snapshots of an "Interactive 3D Print" at one minute intervals. The model starts as a perfect cylinder, but the user can sculpt its shape concurrently with the printing process.

Figure 61 - Digitally sculpted model vs. printed one.

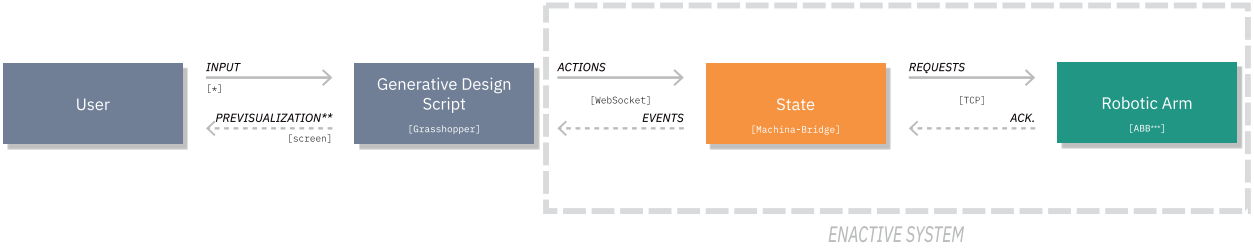


Figure 62 - System architecture of "Interactive 3D Printing" and "Interactive Hot Wire Cutting." In the former, system input consisted of keyboard interaction, in order to drive an ABB IRB 140***. In the later, a game controller* was used to drive an ABB IRB 6700***. In this case, the core script offered no previsualization** of the effects of the input.*

The material research in clay extrusion conducted as part of the "Spatial Print" series also served as the basis for a parallel project, focusing specifically on the capacity of the framework to enable ***Interactive 3D Printing***. The main idea behind this experiment was to explore 3D printing as a potentially interactive fabrication technique (Willis et al. 2011).

Conventional 3D printing workflows are fundamentally based on the offline model. Typically, the designer uses some form of "slicer" software to turn a digital 3D model into a machine program, typically written in G-code, executing it later on the 3D printer. Unfortunately, the role of the designer ends as soon as printing begins, since these systems do not allow user intervention during the fabrication phase. This prevents making on-the-fly design decisions that are informed by the material and fabrication processes.

The goal of this project was to create a robotic system capable of 3D printing clay objects from an initial base model, accepting modifications in the predetermined toolpath by a human on the loop during execution (Figure 62). The system consisted of a Grasshopper script that, starting from an initial input geometry, generated all the slicing and toolpath logic, and transformed it into an action-based Machina program. The program would be streamed for execution to the Bridge split into layer steps: as a layer was sent, the script would listen to incoming events from the Bridge, and upon completion of the layer, the information for the next one would be streamed. The script would also serve as visualization engine, color-coding each layer on the Grasshopper viewport preview based on print status: pending, currently printing or printed.

The contribution of this project was the capacity for the designer to use the Grasshopper script to modify the shape of the object concurrently during the printing process. The script incorporated keyboard input, which allowed the user to traverse a virtual cursor through the printing layers. Further keyboard input would allow the user to expand or contract the chosen layer, except for those layers that the system has flagged as printed or currently printing. Resizing of the current layer would be capped to a safe maximum radius over the one directly below, in order to prevent failed prints. Additionally, modifications on the current layer would propagate to the ones above it, creating a cascading modeling effect in line with the material logic of a 3D print (Figure 60 and Figure 61).

The system proved successful in providing an interactive 3D printing experience, allowing users to dynamically model ceramic objects during fabrication. It provided a systematic way of capitalizing on the precision of the mechanical fabrication process, while incorporating a human in the system as a concurrent decision-making agent[28]. Furthermore, the ceramic medium proved a particularly good fit for this test, as the properties of the material allow printing larger objects in minutes rather than hours for similar pieces using fusion deposition modeling (FDM), a more appropriate time scale for humans on the loop (Figure 62). This project showcases the capacity of the Machina framework to generate a larger spectrum of solutions to supersede traditional fabrication workflows and provide novel and deeper robotic experiences.

[28] "Here, in the lathe or the drill, one has the accuracy of the finest machine coupled with the skilled attendance of the workman" (Mumford, 10).

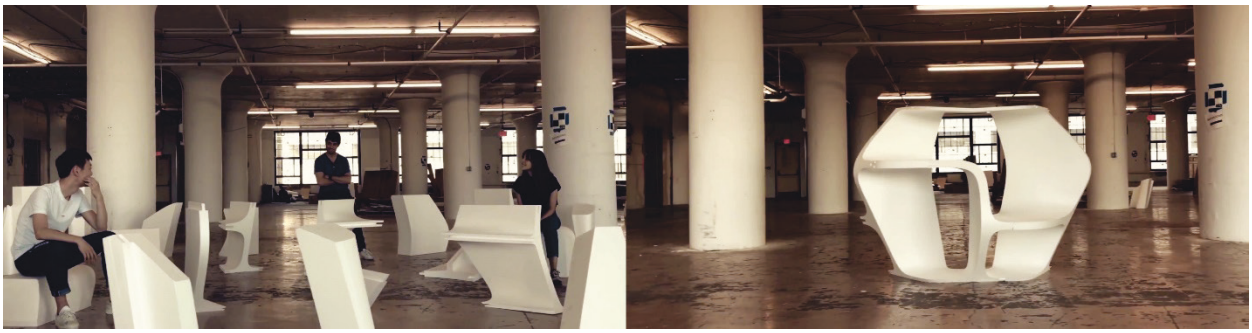


Figure 63 - Hot wire-cut foam "pod" in the "Robotic Hot Wire Cutting" project.

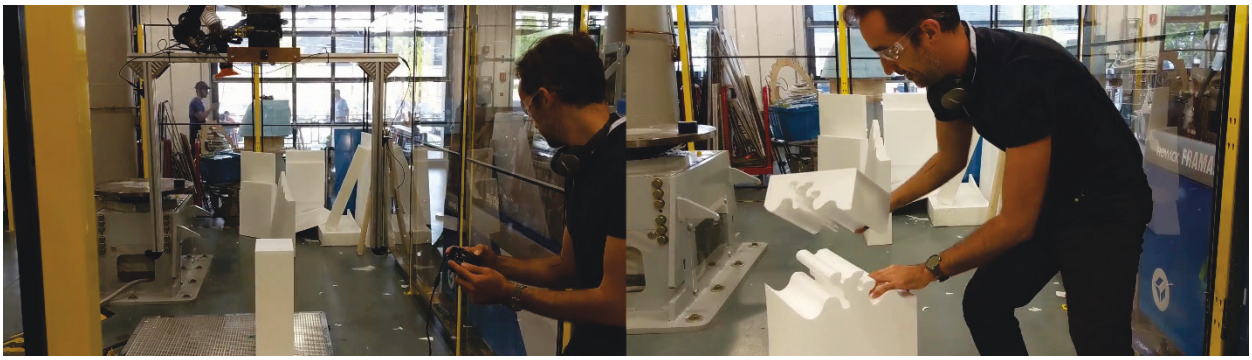


Figure 64 - User controlling the cutting tool in real time with a game controller in "Interactive Hot Wire Cutting."

A similar approach to interactive fabrication was implemented in the ***Interactive Hot Wire Cutting*** project (Figure 64). The project stemmed from the research of the authors in ***Robotic Hot Wire Cutting*** during their residency at the Autodesk BUILD Space in Boston. In this work, Machina for Grasshopper was used to fabricate foam prototypes of inhabitable units (Figure 63), using the technique of hot wire cutting with a tensioned steel wire mounted on an aluminum frame attached to an industrial robotic arm[29]. While concurrent communication was used to streamline toolpath simulation checks on RobotStudio, most of the production work was done offline. This led to speculation on possible avenues to enhance the process, incorporating real-time user input in the workflow.

In this case, the Machina framework allowed the implementation of a rather simple architecture of directed motion control. A Grasshopper script was created, which used the Xbox-Grasshopper plugin to continuously read input from a video game controller. This input was directly translated to relative actions and streamed to the robot. This allowed a human on the loop to enact direct control over the motion of the hot wire, guided by the live feedback gathered from the system itself. As opposed to "Interactive 3D Printing," user input was not bound to a baseline pre-programmed toolpath. Instead, the user had all degrees of freedom available for robotic motion, with the system constituting a sandbox environment for robotic sculpting of foam (Figure 64). The material medium also proved appropriate for this real-time application, as the slow speed required to cut foam with a hot wire is well suited for decision making and a safe human-robot interaction.

[29] This research was later extended into the ***Future of Living*** project (see Appendix A: Project Credits), using a similar technique to generate custom formwork for concrete pouring.

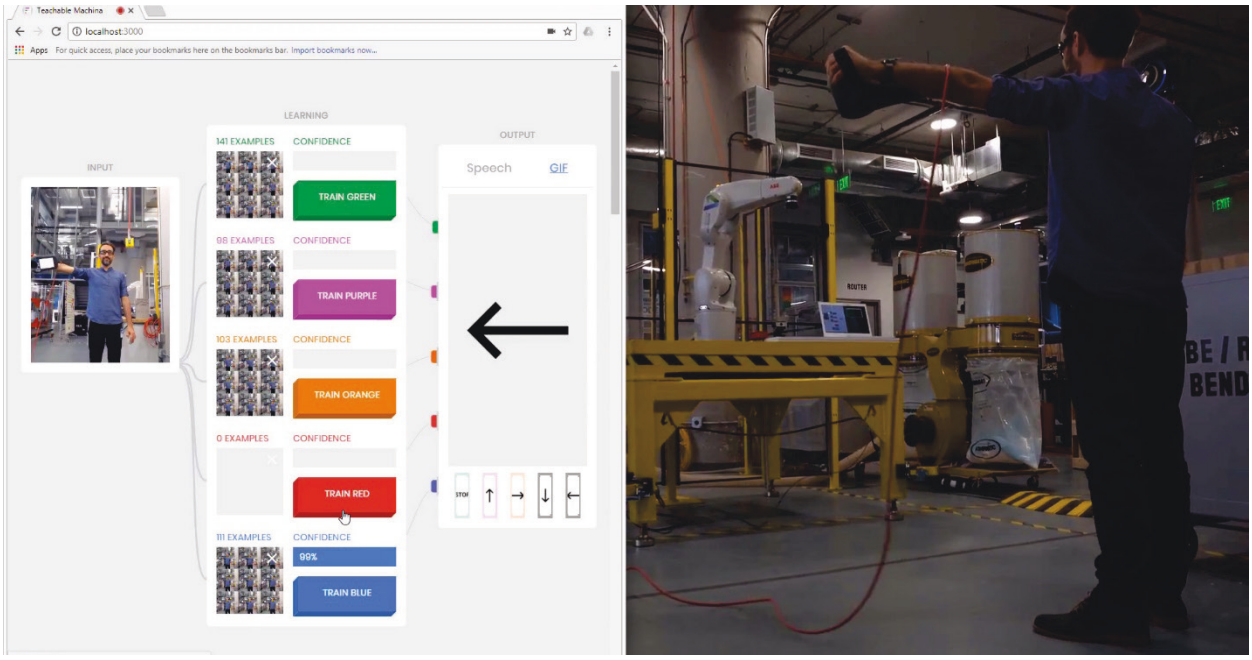


Figure 65 – The "Teachable Machina" interface, with the user training the model to jog the robot in Cartesian space.

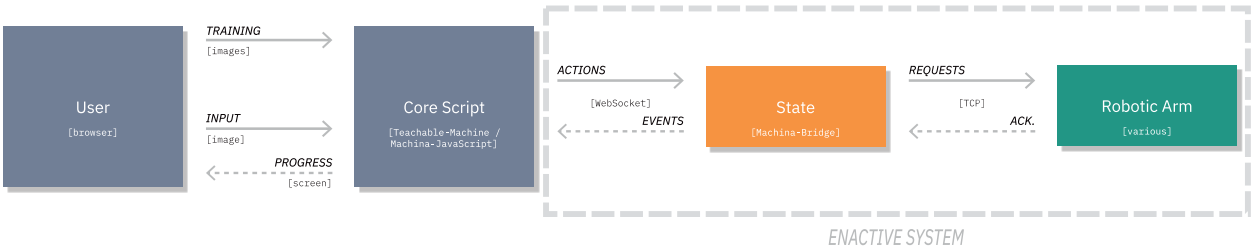


Figure 66 - System architecture of "Teachable Machina."

The potential of incorporating machine learning as decision-making agents in robotic fabrication projects was also explored in the development of ***Teachable Machina*** (Figure 65) and its sibling projects.

The project was built on top of the `Teachable Machine` framework, "an experiment that makes it easier for anyone to explore machine learning, live in the browser – no coding required" ("Teachable Machine" 2017). This open-source project provides the possibility of training a neural network to classify images captured from a webcam. The unique characteristic in this project is that such training, which would typically take hours and require massive amounts of data, can be done live on a browser in a significantly reduced amount of time — usually a few seconds, depending on the host computer. The pure browser-based nature of the project makes it also quite convenient for novel users, as it requires no downloads, installation or coding, providing a simple, immediate and fluid interface[30].

The authors adapted the project and used it to create "Teachable Machina", a trainable tool for real-time, image-based robot jogging (Figure 66). The original framework was enhanced to accept further input classifications, and the outputs where replaced with icons corresponding to four Cartesian directions plus a stop sign. Additionally, the framework was also extended with the Machina for JavaScript library, in order to interface with the robot via the Bridge. The output of each one of the input classifications was linked to a simple relative `Move` action, and streamed to the robot. Feedback from the state of machine execution was used to properly synchronize the requests for motion. To avoid overflowing the robot with actions requests at the webcam's refresh rate, the system would issue a request, and wait until its completion to issue the next one, according to the user's most recent input image. This architecture resulted in an interactive interface where users could train the system to associate certain images —body poses, facial gestures, colors, objects— with robotic instructions, being able to control the movement of the robot by enacting the corresponding image in front of the webcam. As with "Interactive Hot Wire Cutting", the system provided a full range of unbound motion, without the need for any baseline program.

[30] More info about the project can be found in Webster 2018.

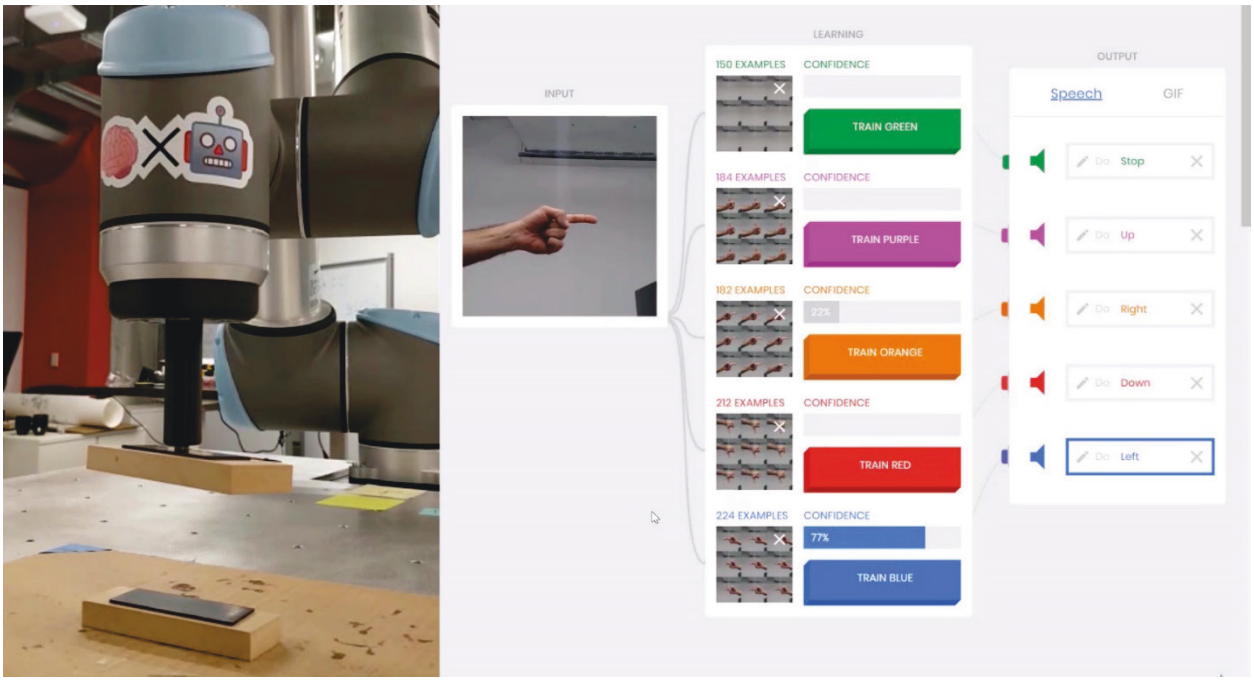


Figure 67 - In "Mediated Construction" users could train a neural network on site to recognize body gestures and link them to specific robot procedures associated to stacking a pile of bricks.

Figure 68 - In "Slip Cast Turntable" a robotic arm performed a continuous rotation of a mold containing slip, accepting user commands in order to tweak the rotation angle.

"Teachable Machina" was used as the base template for two projects developed by participants of the Mind Ex Machina workshop. In ***Mediated Construction*** (Figure 67), the authors designed an interactive robotic pick and place interface, in order to fetch small wooden planks from a known location, and stack them on a "Jenga-like" tower.

The pick and place routine was pre-programmed with the underlying Machina for JavaScript library as the baseline operation in the system. Significant stages of the full program were divided between the different output categories: pick object, move to approach position and drop piece, with the system keeping track of the number of stacked pieces and adjusting the program correspondingly. Additionally, two other inputs were dedicated to unconstrained spatial rotation of the pieces, allowing the user to customize the orientation of the plank prior to placement. The sequencing and triggering of these programs could be controlled on the browser by the user through training the inputs with custom images on the webcam.

The ***Slip Cast Turntable*** project constituted an interactive robotic experiment for the crafting of ceramic slip cast objects (Figure 68). Similar to "The Mixology Table," the project aimed to automate the tedious process of rotating a mold to distribute the slip homogeneously. However, in this case, the authors chose to incorporate a human crafter in the system in order to control the angle of the rotating bowl as the slip solidified. For that purpose, an infinite loop program was implemented in the "Teachable Machina" framework, making the robot rotate the bowl mold in cycles of 360 degrees. Additionally, two inputs where programmed to modify the vertical rotation angle of the bowl in positive and negative increments. The user could train the model to perform such rotations responding to particular gestures, and therefore, be in fine control over the distribution of the material over the mold's surface.

Both projects resulted in robotic systems controlled by the user though interaction with a neural network trained on the spot. In a similar way to "Interactive 3D Print", the user had control over the flow of a baseline program while, at the same time, allowing a certain degree of customization through small, open-ended procedures. In the case of "Slip Cast Turntable," the project evoked a new form of robotically-enabled digital craft, using machines to relieve the artisan from tedious repetitive tasks, and allowing them to concentrate on the creative process. In the case of "Mediated Construction," the interaction paradigm served as speculation on the future of robotically-assisted construction, with machines performing baseline automated building tasks guided by the decision-making capacity of humans on the loop. Both projects manifested the capacity of the Machina framework to provide an architecture where implementation of concurrent interaction with a robot control system was possible natively and effortlessly.

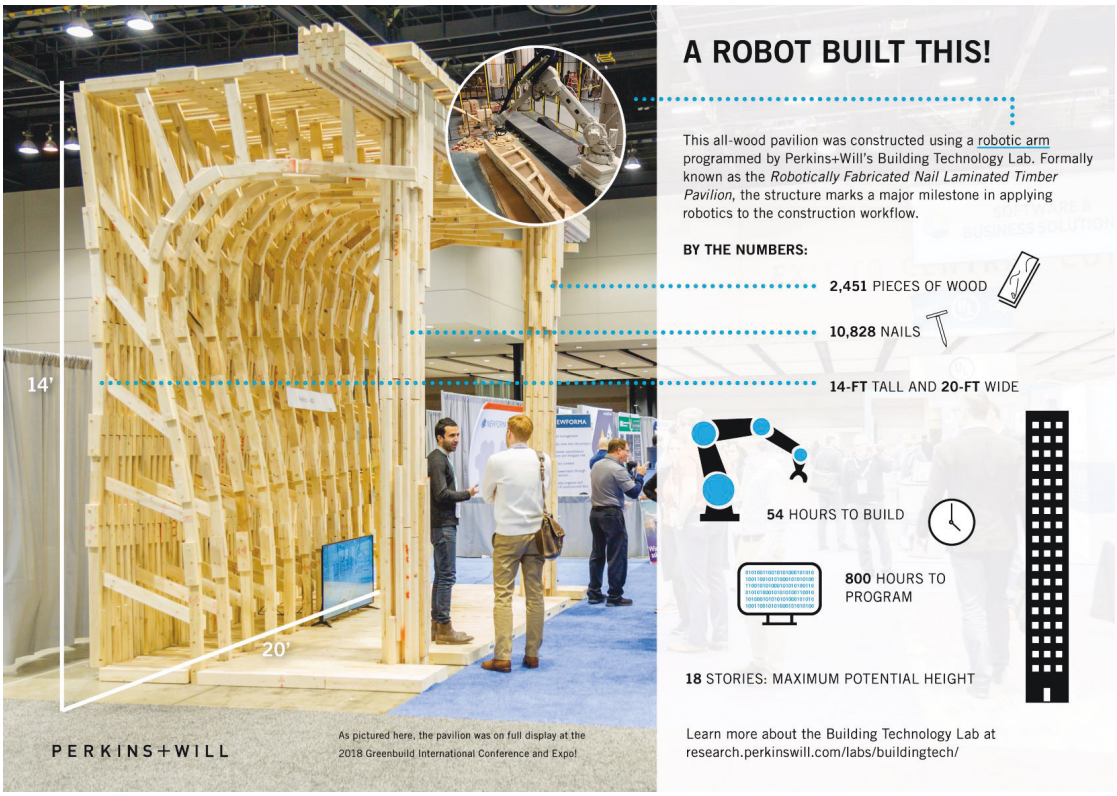


Figure 69 – Marketing material from the "Greenbuild Pavilion" project.

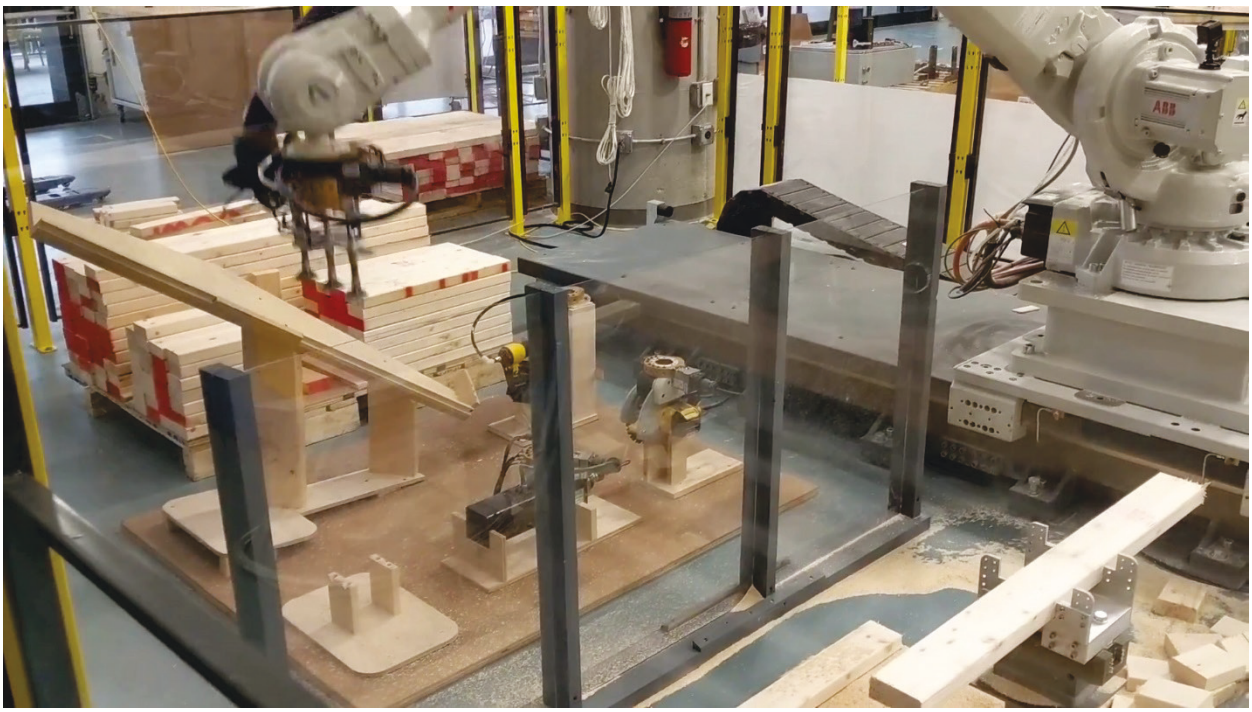


Figure 70 – Fabrication setup on the "Greenbuild Pavilion" project, including material stock, tool-changer platform, and cutting jig.

5.1.3 Building

The projects discussed previously in this section showcase different architectures for the development of robotic systems, allowing a wide variety of interaction models at different levels of concurrency. However, the one common thread between them is the relatively constrained range of their motion and the reduced scale of their outcomes —mostly objects that can be easily held or manipulated by a human. This still leaves the open question of the potential benefits of concurrent robot control mechanisms applied to larger scale tasks such as, for example, building and construction projects.

The possibility of testing the suitability of the Machina framework to enable large scale construction projects is especially appropriate in the context of this dissertation for two main reasons. On the one hand, the bigger scale and workspace volume of industrial robotic arms makes them a particularly good fit for building-scale manufacturing, as compared to other CNC machines. On the other, the real-time connection to decision-making agents enabled by the enactive robotics model may provide a fertile ground for the development of systems that address the elevated degrees of uncertainty present in construction sites. The model could allow the systematic development of smart control mechanisms that provide flexible and adaptable solutions for on-site robotic construction (Bechthold 2010).

One of the earliest adopters of the Machina framework was the Building Technology Lab from Perkins + Will, in application for the construction of their ***Greenbuild Pavilion*** (Figure 69). In this project, a large industrial robotic arm mounted on a 30' long track was used to fully automate the construction of a 20x14x6' wooden structure.

A full parametric modeling script was developed for the generation of the pavilion model, with the design specifications customized to the individual member level. The script used the Machina for Grasshopper plugin to generate the action-based full sequence of robotic instructions, in order to generate the pavilion modules from 2x4" timber frames. The assembly sequence included picking up an untrimmed stud from a pile, placing it on a jig, cutting it to measure with a circular saw, placing it on its final position, and using a nail gun to fix it to its neighboring members. The program also incorporated the necessary routines to attach and detach dedicated tools for each operation (Figure 70), as all the tasks were performed by the same single robotic arm. The fabrication process was executed offline, fabricating the structure over the course of 54 hours, using approximately 2500 wooden frames, 11000 nails and two supervising humans. The pavilion was on exhibit during the 2018 Greenbuild International Conference and Expo in Chicago.

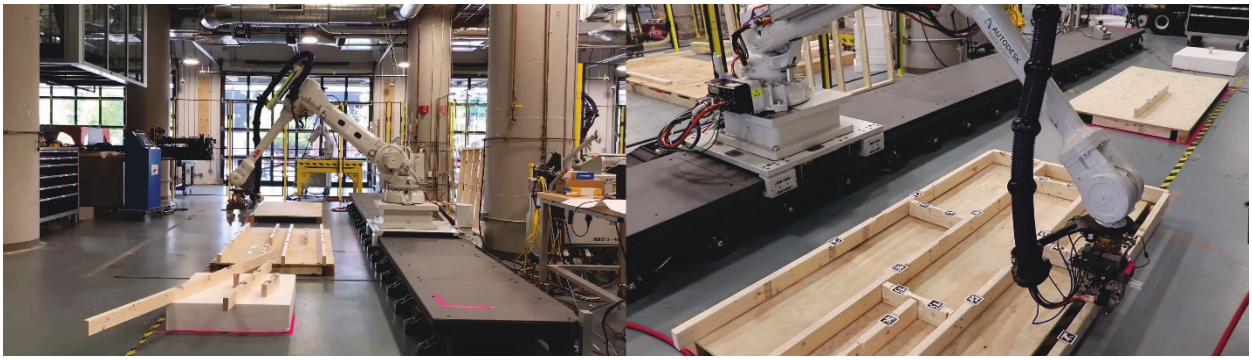


Figure 71 - In "Automated Fabrication" a computer vision system helped the robot locate the uncalibrated position of construction members and place them ready for assembly.

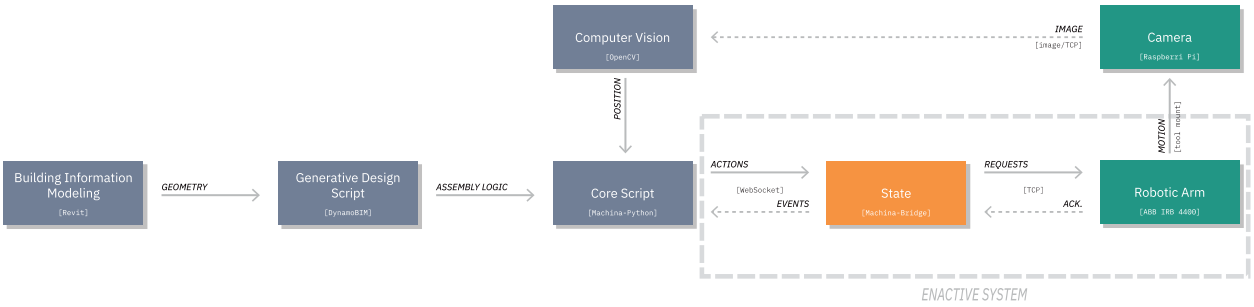


Figure 72 - System architecture of "Automated Fabrication."

The "Greenbuild Pavilion" project demonstrates the robustness of the Machina framework to provide a reliable, production-grade solution for large scale robotic construction projects, using conventional offline programming. This was made possible by the fact that the pavilion was fabricated at the Autodesk BUILD Space in Boston, in a controlled environment with restricted human access, specially trained personnel and a highly calibrated setup. However, this is the opposite scenario to a typical construction site, where machines and humans are in constant flux on a shared space, building tasks must dynamically adapt to a rapidly changing environment, and digital to physical consistency cannot be guaranteed.

The ***Automated Fabrication and Assembly in Construction*** project (Figure 71) constituted an attempt to speculate on possible solutions to address the uncertainty typical on construction sites, by using real-time sensor feedback from the environment on a large assembly process.

The project started from a building information model (BIM) in which a conventional wooden balloon frame wall panel was created. The authors used generative design tools to automate data extraction for CNC fabrication of the individual members. Each member was augmented with the addition of two fiducial markers, in order to make them identifiable and trackable. The robot arm was equipped with an industrial gripper, to which a camera was attached. The camera provided a video stream to the CV module integrated in the system. The real-time position of the robot was used to compute the orientation of the camera, and in tandem with the CV input, the location of the markers in 3D space (Figure 72). This allowed the system to recognize which members were available for assembly, estimate the gripping point, and generate a procedure to pick up the frame and drop it on the right location on the assembly surface. The procedure would be streamed to the robot for execution, while the construction worker would place another frame on the pick-up bench, with the process starting over again.

This prototype explored the potential of using computer vision and real-time robot control on a construction site to automate assembly processes. In this scenario, precision work is shifted from the site back to the factory, where the technical conditions are better suited for this purpose. A batch of "smarter" objects could be deployed on site, and unskilled personnel could stack the elements on a delivery platform, bypassing the need for careful arrangement and sorting of the elements. Avoiding the requirement of on-site calibration constitutes an important step forward towards a full implementation of robotic construction, satisfying the requirement for adaptability of the system, and helping to compensate for human error.

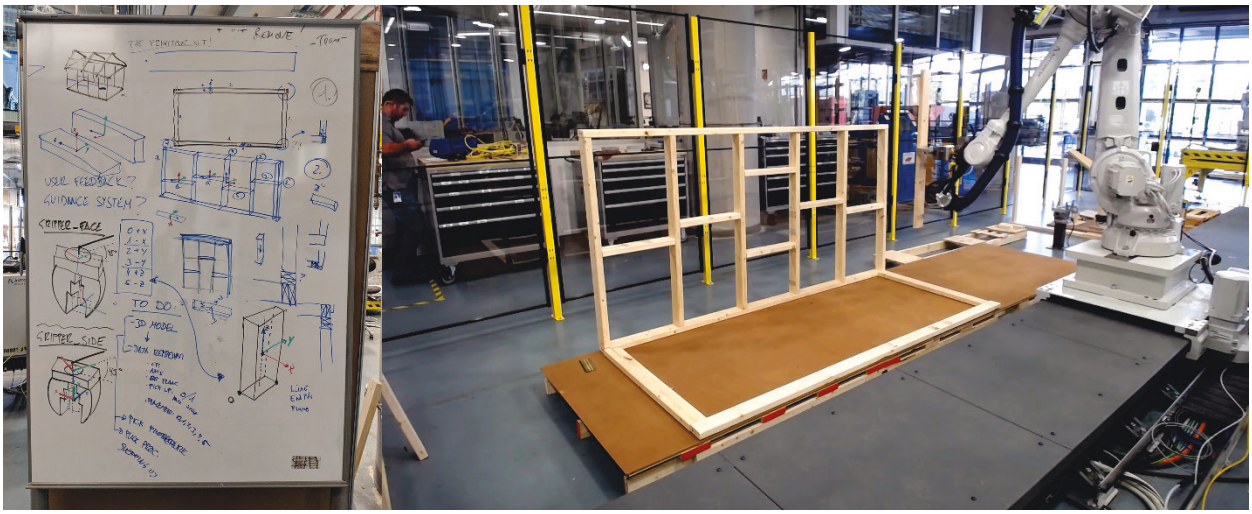


Figure 73 - Images of the "Primitive Hut" project, with model planning in relation to robotic construction (left) and first erected wall (right).

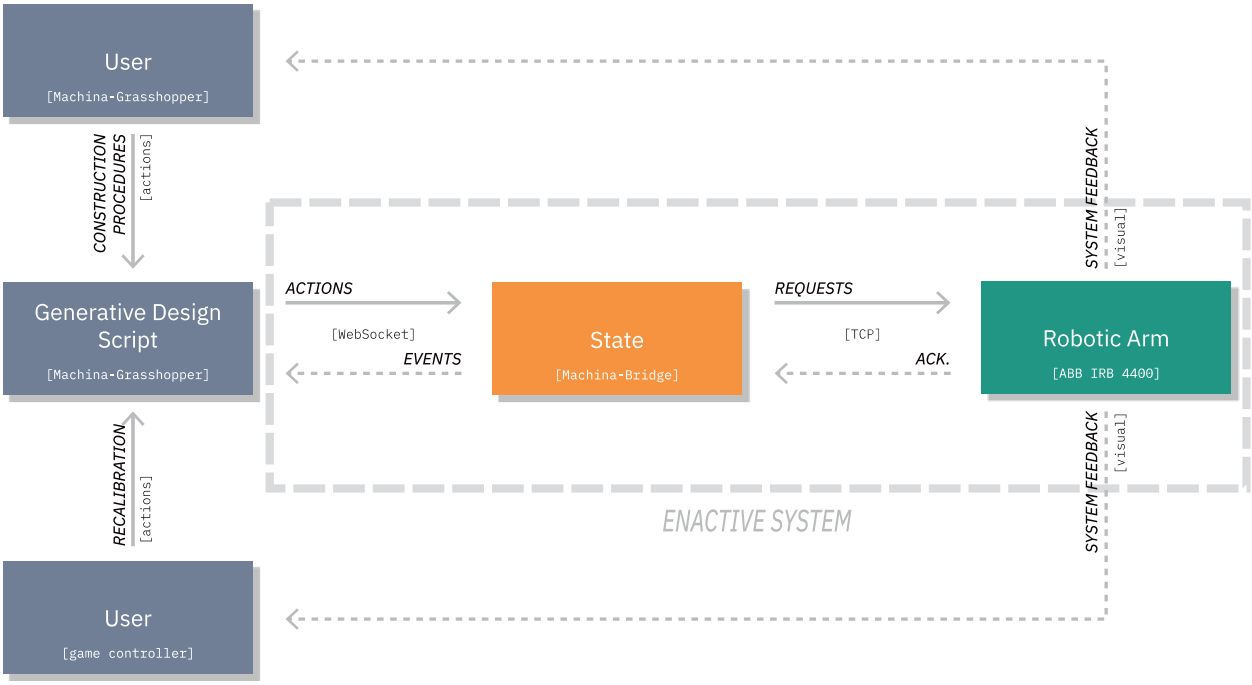


Figure 74 - System architecture of "Primitive Hut."

The notion of real-time control as measure to create more adaptable robotic systems was further explored on the ***Tight Squeeze Pavilion***, built as part of the Tight Squeeze workshop at the 2018 RobArch Conference (Figure 75). In this project, the main goal was to incorporate humans on the loop as part of the robotic construction process, in order to perform recalibration tasks and possibly design decisions.

As in "Automated Fabrication," the authors started with a digital 3D model incorporating a full description of the pavilion design at the member level. In this case, each frame allowed the additional introduction of certain metadata parameters, such as internal orientations and type enumerators, in order to allow the designer to customize member properties related to its robotic assembly —such as where the element would be picked from and how it would be attached to the rest of the structure. A master script was designed in Grasshopper to create action-based pick and place programs for each one of the members, broken down into sequential procedures: approach frame, pick up, approach drop location, open gripper, etc. Each one of the procedures could be streamed independently to the robot, via the Machina Bridge, allowing a human controller to issue the program sequentially with a control panel-like interface. Additionally, the system included real-time input from a video game controller, associated to simple relative motions in Cartesian space. This controller could be used by another human to perform small jogging operations in between procedures, in order to readjust positions or make small tweaks.

This system was used to build a prototype named ***Primitive Hut*** (Figure 73). The model was a simple 8x4x4' box built with 2x4" studs in the balloon frame system. As opposed to traditional balloon frame, were walls are assembled flat on the floor and erected as a continuous element, the goal of this project was to be able to assemble all the elements of the wall directly in their final position. Individual elements were precut to measure, and placed on a calibrated jig for robotic pick up.

The contribution of this project relied on the different possibilities for direct control embedded on the digital interface, which enabled a fairly novel interaction model in robotic construction. The system allowed one human to become the "master controller," with direct control through Grasshopper over the choice of member to place, and which step of the program to enact; pressing on dedicated UI elements would stream the actions corresponding to different steps of the program, allowing them to cycle through the pick and place operation, go back and forth, or jump to a different stage if necessary. The nature of the Grasshopper interface allowed the controller to make changes and improvements to the code while concurrently controlling the machine, in line with the main ideas behind the enactive model. However, the material tolerances in the construction process often made members tilt or warp, causing elevated mismatches between the digital model and the constructed object. For this matter, the role of a second "on-site controller" was necessary in the process. This agent had the capacity to use the game controller to move the robot in small increments, in order to perform on-site recalibration and adjust the location of the member according to the warped physical object (Figure 74). The capacity of this controller to perform unbound robotic actions not prescribed in the baseline program allowed them to further extend those actions to include on-the-fly design changes informed by the state of the construction. Upon placement of an element, the system would be notified, and the digital model will update

the location of the placed member, forming an "as-built" model of the final outcome. Communication between controller agents would allow them to coordinate construction efforts and make informed decisions collaboratively[31].

Figure 75 - "Tight Squeeze Pavilion:" an on-site controller was able to perform changes over the baseline robotic program and plan drop-off motion (left), resulting in an intricate assembly built in three days by two robots and four human supervisors (right).

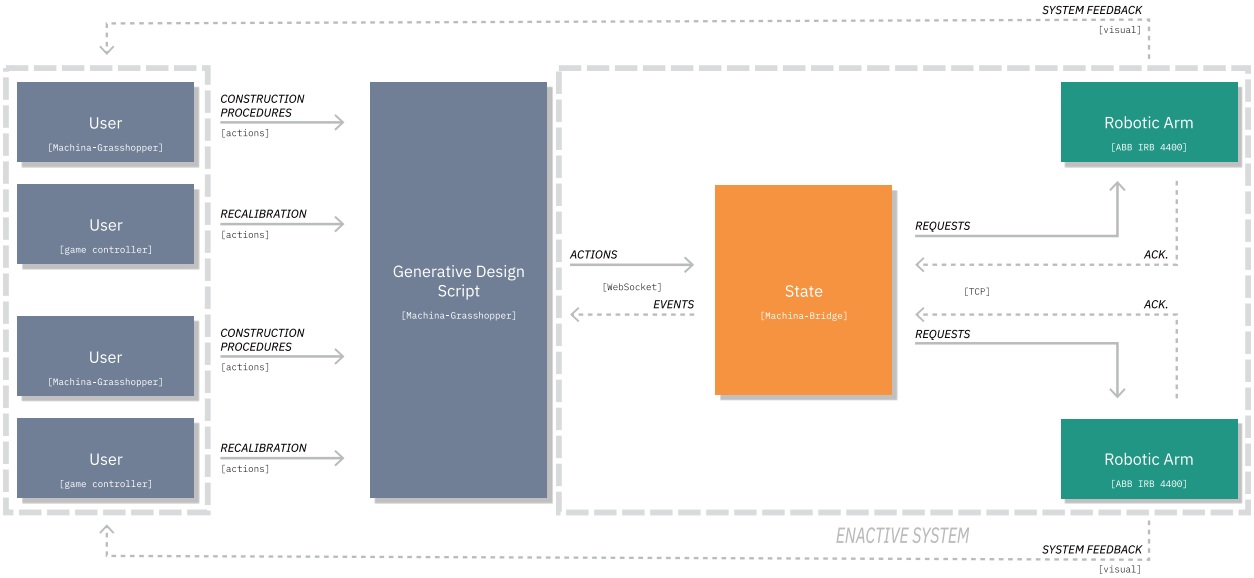


Figure 76 - System architecture of "Tight Squeeze Pavilion."

[31] Two human controllers were part of the construction of the "Primitive Hut" project. However, this was done for the sake of speeding up the process; nothing in the architecture of the system would prevent a single agent to acquire both roles.

The interaction model put forth by the "Primitive Hut" prototype was further extended into the "Tight Squeeze Pavilion." The workshop was themed around the idea of constrained construction sites, and the limitations this could impose on the design of buildings to be executed by robots. As a result, the design of the pavilion was purposefully intricate, requiring the development of specialized solutions to deal with the complexity of the context (Figure 75).

The solution proposed in the workshop was to enhance the capacity of the system to respond to these constrains by harnessing the decision-making capacities of the humans on the loop. The system was adapted to build the pavilion with two robotic arms hanging from a gantry (Figure 76). A team of four workshop participants acted as the control system in charge of the successful construction of the designed pavilion. Furthermore, the "on-site controllers" were given enhanced real-time jogging capacities, embedding more complex spatial motion capacities within their game controllers. This allowed them the possibility to perform "live" motion planning by performing guiding operations between the pick-up and drop stages, according to the current state of the built object. Moreover, the system did not include any coordination events or collision checks between the two robots; these tasks were also managed by the human controllers with the help of the concurrent control interfaces.

The "Primitive Hut" and the "Tight Squeeze Pavilion" highlight the potential of the Machina framework, and hence the Enactive Robotics model, to systematically enable deep concurrent human-robot collaboration. In scenarios where the complexity of the context in which the robot must work goes beyond the capacity of the programmer to anticipate them, the possibility of incorporating humans as an integral part of a robotic system becomes a great asset. In such collaboration, the power of the robot to perform precise and tedious repetitive tasks can be coupled with the flexible capacity of human brains to make adaptive decisions based on changing contexts.

5.2 Discussion

The case studies discussed in this section showcase the wide range of possible robotic architectures enabled by the Machina framework. The purpose of this elaboration is to demonstrate how concurrent robot control systems, built after the principles behind the Enactive Robotics model, can systematically provide a richer and deeper spectrum of interaction paradigms than traditional robot programming systems.

A descriptive introduction of each case has been presented, as well as comprehensive characterizations of the parts in the architecture of each system, the relation between them, typical use case scenarios and the interaction models thereby suggested. The projects have been structured in relation to the nature of their outcomes, as well as how the findings in a particular instantiation informed the course of the next iteration; the reader is invited to speculate on new threads to intertwine them. As a group, they contribute to advance

the understanding of how these new paradigms can affect the way typical robotic processes are implemented, such as drawing, making or building.

However, the discussion around some of the most relevant underlying aspects behind these projects in relation to this dissertation —the ones relating to the postulates of the Enactive Robotics model— has been deliberately postponed after their introduction, in order to gain group perspective for their analysis.

5.2.1 Skill

The varied background and skillsets of the authors involved in the creation of these projects evidences the capacity of the Machina framework to provide a unitary solution space, able to cater to the requirements of all different levels of user expertise.

In principle, the framework is designed to provide an immediate and accessible entry point to robotics for novel users. Machina relies on the action model to provide a simple API of verb-based instructions to prospective users, which relates to a general, platform-agnostic understanding of spatial motion and machine control. This model is in contrast with most robot programming applications, which require previous domain expertise in order to access their functionality. This was manifested in *Material Systems*, where a group of design students with beginner-level technical expertise and no prior robot experience, were able to design and implement complex robotic procedures for the digital fabrication of ceramic objects.

However, this experience also highlighted the cognitive challenges behind offline-based robot programming; the detachment from the machine during the learning process resulted in the average student requiring significant abstract training, and even after it, many still expressed high degrees of uncertainty and lack of confidence when approaching the machine for the first time.

The addition of the Bridge to the Machina ecosystem, and its systematic use as the main introductory platform for novel users, became an incredible leap forward towards a more intuitive learning model for machine control. The concurrent nature of the interaction postulated in the enactive model proved a solid foundation for the cognitive processes involved in understanding the robot. Instead of undertaking long sessions of abstract training, the application allowed users to set up a live connection to a real or virtual robot in a matter of minutes, and start streaming actions in real time. The possibility of typing an instruction and witnessing its effect on the machine right away constituted a vehicle to action-based learning, fostering the immediacy of the physical medium as an affordance towards its comprehension and control. The state-based logic proposed by the enactive model also contributed to the user's understanding of the cumulative effect of relative actions, supporting a structured approach to sequence-based instructing. Training sessions lead through guided robot

interaction allowed constructivist learning approaches, which ultimately led to faster and more solid foundational knowledge[32].

The success of this learning model was manifested by the capacity of many of the authors of these projects to learn and develop complex robotic systems in fairly reduced timeframes. Such was the case for the team behind *Robotic Hot Wire Cutting*, who were the first users to receive Bridge-based training, and who were able to learn, develop and fully produce their prototype in a two-week timeframe. Similarly, the participants in the Tight Squeeze workshop were able to learn robot control in a morning session, and immediately become active programmers and controllers of the two robots used for the *Tight Squeeze Pavilion*. Ultimately, this notion is illustrated by the wealth of different interactive robotic projects and their varied nature developed in the Mind Ex Machina workshop. In a four-day timeframe, participants were able to learn and develop advanced projects involving robotics and machine learning, such as *Selfie Plot*, *Exquisite Corpse*, *Mediated Construction* or *Slip Cast Turntable*.

Furthermore, the Machina framework is also designed to provide a deeper scope of possibilities in robot control for advanced users; after all, the hope is that most novice users will eventually become so. The concurrent paradigm proposed at the core of the enactive model, enabled by real-time communication and bidirectional streams of data, is specifically designed to supersede obsolete offline programming systems, providing a wider range of possibilities and fostering enhanced models of robot control. The ultimate goal of the Machina ecosystem is to enable this possibility systematically, natively and without the need for ad hoc solutions. Furthermore, the modular architecture of the applications in the Machina framework allows the extension of the logics of the enactive model to virtually any digital environment; advanced users with the technical capacity to implement these connections can rely on the framework to manage their need for real-time control, and instead focus on the creative task at hand.

This capacity is particularly illustrated by those projects developed by users with a more technical background. The *Greenbuild Pavilion* demonstrates the capacity of an expert user to use the Machina framework for a large scale, production-level construction project; the fact that the building was produced with offline programming exemplifies how this paradigm is a one-directional, strict subset of the possibilities enabled by the model. *Real-Time Anatomy* showcases the advanced capacity of the users to build a complex architecture of modular processes in order to harness the real-time stream of bidirectional data in the system. Moreover, *Dereliction* proposes a novel application of real-time robot control for simulation and optimization purposes. And lastly, the series of robotic 3D printing examples described in *Responsive Spatial Print* and *Interactive 3D Printing* challenge the unidirectional methods employed in standard 3D printing, and expose alternative techniques to close back the loop, establishing material and user-informed dynamic fabrication workflows.

[32] These ideas are further elaborated on the "Controlled User Study" chapter.

5.2.2 Agency

Typical robot control mechanisms require all decision making to be embedded a priori in the system. The deferred nature of offline execution, isolated from the environment where the program originated, requires self-contained algorithms for control logic, which can only respond to situations accounted for on program creation. The elevated level of anticipation required in the design of such programs poses a great challenge, even for the most skilled programmers and experienced roboticists. But even in this case, the closed nature of these hermetic regulation mechanisms becomes a constraining platform for their agency, preventing the adaptability, flexibility or dynamic improvement of robotic systems.

The Enactive Robotics model situates decision-making agents at the forefront of robotic systems. The capacity of any agent, regardless of its nature, to interact with an enactive system via a standard of actions and events, allows a system architecture that shifts all the low-level machine control management to the core state model, leaving regulation to logic units connected concurrently to it.

The simplest form of logic driving an enactive robotic system can be constituted by a fundamentally deterministic algorithm. This is the case in comprehensive fabrication projects such as *Material Systems* and the *Greenbuild Pavilion*, where the core program was in charge of the translation between complex geometries and robotic actions, without the need to incorporate system feedback into the workflow. As with offline systems, unidirectional pre-canned logic is also a strict subset of the possibilities within the Enactive model.

However, deterministic algorithms may also take advantage of the closed-loop architectures enabled therewith. The *Responsive Spatial Print* project used sensor data as a self-regulation mechanism, in order to recalibrate the geometry of new layers according to the measured deformations of the preceding ones. Similarly, the control mechanisms designed in *Dereliction* for heuristic toolpath optimization required system feedback to detect invalid robot poses. Lastly, computer vision was used concurrently with robot control in *Automated Fabrication* to detect frame members and create on-the-fly pick and place procedures. While the projects were driven by reasonably simple sets of rules, concurrent communication and system feedback provided by the Machina framework were critical to achieve the successful results of these projects.

The independency of regulatory mechanisms in Enactive Robotic systems makes any unit with capacity for input/output processing suitable as a decision-making agent, regardless of its nature. This includes more complex forms of regulatory algorithms such as, for example, machine learning models. The use of artificial intelligence in robot control mechanisms is not a novel technique. However, the elevated technical requirements necessary to use them in real-time operations is usually beyond the capacity of the machine's microcontrollers, requiring dedicated equipment and customized solutions for cross-platform communication.

One of the advantages of the Enactive model is to offer a systematic solution to this problem. The conceptual architecture proposed by the model requires decision-making agents to be independent from mechanical controllers, and provides a communication protocol to interface with them. This streamlines the incorporation

of machine learning models as regulatory mechanisms for robotic systems. Such feature is demonstrated by the *RobotSketch-RNN* and *Exquisite Corpse* projects, using neural networks to suggest collaborative robotic drawings with human users, or the series of projects stemming from the work on *Teachable Machina*, allowing users on-site training of machine learning models to aid on different robotic tasks.

Ultimately, it could be argued that one of the most elevated forms of decision-making agents are humans themselves. The capacity of the Enactive Robotics model to provide a systematic way to incorporate humans on the loop opens the door to the creation of truly human-robot collaborative systems, combining the power and precision of robotic machines with flexibility and adaptability of human brains. This was the case for example on *Drawing with Robots* or *Selfie Plot*, where users were given the possibility to interact and control a playful robotic system for live drawing. Similarly, the *Real-Time Anatomy* project allowed a human controller the choice of projection angle, in order to augment the body of a passive subject with an overlay of anatomical information. Lastly, the *Interactive 3D Printing* and *Tight Squeeze Pavilions* projects featured robotic fabrication workflows designed to incorporate human decision-making —or even teams of them— at critical stages of the baseline programs.

It is important to highlight that in the majority of these projects, regardless of the perceived leadership on the system, the decision-making mechanisms are driven by a hybrid combination of different agent types. Most projects incorporate some form of deterministic logic, either as data processing mechanisms or as mediators between neural networks, humans, the core state model or all the above; the *RobotSketch-RNN* project is noteworthy in this sense. The line that separates the different parts of these systems is usually rather blurry, either from a technical, interactive or conceptual point of view. The capacity of the Enactive model to foster hybrid control mechanisms with modular architectures of multiple agents is key in providing a fertile foundation for the creative exploration and speculation on novel interaction paradigms for robotic systems.

5.2.3 Concurrency

The Enactive Robotics model is founded of the capacity of the decision-making agent to concurrently enact control over the mechanical counterparts. This typically involves the exchange of information in the form of action requests and event listening, and the capacity of the agent to link the channels through logic structures. Nevertheless, even though the model conceives communication mechanisms to happen in real time between the parts, the rate at which this interaction flows is left at the discretion of the system designer.

The projects show the capacity of concurrency in an enactive system to be understood as a gradient of interaction levels, rather than a binary form. Certain tasks may require extensive preplanning and data generation, but once computed, a full program formed by thousands of actions may be streamed to the robot, with no need to further account for any system input or react to its state. This is the case in typical CAD/CAM workshops, where intense geometry processing algorithms lead to the post-processing of fabrication routines,

which require no system feedback for their execution. This was the case for instance in most of the *Material Systems* projects, as well as the *Greenbuild Pavilion*. In this sense, the focus on directed, feedback-less control of the machine can be seen as the lowest form of system interaction —akin to the one fostered by offline systems, a subset of the Enactive model as mentioned. It is worth noting that, while non-critical for the correct work of the system, the stream of system input is still useful for monitoring, data collection purposes or visualization, such as in *Drawing with Robots* or *Selfie Plot*.

A step further in the concurrency ladder can be found on systems that require state feedback only at significant stages during the execution of a program, or where interaction is reduced to minimal interventions over a baseline behavior. Many of the projects described in this section fall under this category, as most physical robot procedures take significantly longer to execute than the time it was required to compute them (see §4.1.5). In *Dereliction*, the system needed to wait until a toolpath simulation potentially failed in order to flag it appropriately, generate a rearm routine and issue the actions corresponding to the next toolpath. The systems in both *Responsive Spatial Trajectory* and *Automated Fabrication* only triggered their sensing and action-based requests in preparation for the next step of the program —a new layer, a new frame—, remaining idle during its execution. Alternatively, the systems developed for the *Tight Squeeze Pavilion* and the *Teachable Machina* series worked on top of a continuous, pre-programmed routine, prompting the users for input at specific points during execution where some degree of customization was allowed by design. This turn-based approach was especially convenient for human-robot interactive systems, as reducing intervention to punctual moments on a slow-paced program increased the safety around robot manipulation, reduced anxiety on participants and allowed newcomers to familiarize themselves with the interaction model offered by the system.

Finally, perhaps the most elevated level of concurrency could be attributed to those systems where data exchange and decision-making happen at a nearly uninterrupted pace. This was the case in the *Real-Time Anatomy* project, where the system continuously queried the computer vision system for the position of the projection board, trying to maintain robot orientation as close to perpendicular as possible. Similarly, in the *Interactive 3D Printing* project, the user could continuously change the shape of the printed object, with layer progression so fast that program streaming approached real-time speed.

6 Controlled User Study

In the previous chapter, several case studies of robotic applications are described, built with Enactive Robotic control systems. These projects support one of the main theses of this dissertation, evidencing that the model can constitute a framework for creators to develop more complex robotic systems and provide a deeper spectrum of possibilities due to its concurrent nature. The target group for this claim is mainly advanced users who are technically savvy, or who have prior experience with robots.

This dissertation also argues that the Enactive Robotics model offers an additional advantage for novice users over traditional robot programming systems: it systematically provides an easier, immediate and more intuitive entry point to robotics, as the architecture and interaction paradigm fostered by the model aid in the cognitive processes that lead users to understanding the medium. While certain evidence of this claim is provided in chapter 0, the case studies do not specifically address the learning processes that led to their implementation, nor provide quantitative data to characterize them.

As part of the research conducted in this dissertation, a controlled user study was designed in order to test this assumption. The main goal of the study was to observe a group of participants using an industrial robotic arm for the first time, and compare their performance and feedback upon trying two different programming environments: an "easy, user-friendly" programming interface provided by a popular robot vendor, and a control system built after the principles of the Enactive Robotics model. The qualitative and quantitative data gathered from this study was expected to provide evidence to support the main thesis of this dissertation.

In this chapter, the controlled user study is presented, along with an interpretation of the findings and a detailed analysis of the data gathered. First, the methodology of the study is described, including structure and overall guidelines, and the applicability of the chosen method is framed in relation to the comparative nature of the study. A detailed account is given about the chosen robot programming systems and their suitability to serve as a benchmark to evaluate the premises behind this study. A general summary of the findings of the study is offered, followed by an annotated analysis of the data that supports it: user demographics, their

platform-specific feedback and the results of a comparative survey between them. The chapter concludes with some conclusions that highlight the relevance of the study in the scope of this dissertation, and manifest certain challenges identified as a consequence of this experience.

6.1 Methodology

6.1.1 Structure

The proposed study was conducted using the crossover model. A crossover study is one where participants are exposed to different environments sequentially, and where results are compared between groups where the order of this sequence is randomly changed. The main affordance of this experiment model is to compare the effects of all the environments on the same subject, while being able to identify —and potentially isolate— the carryover or residual effects of the previous exposure to the next (Jones & Kenward 2014).

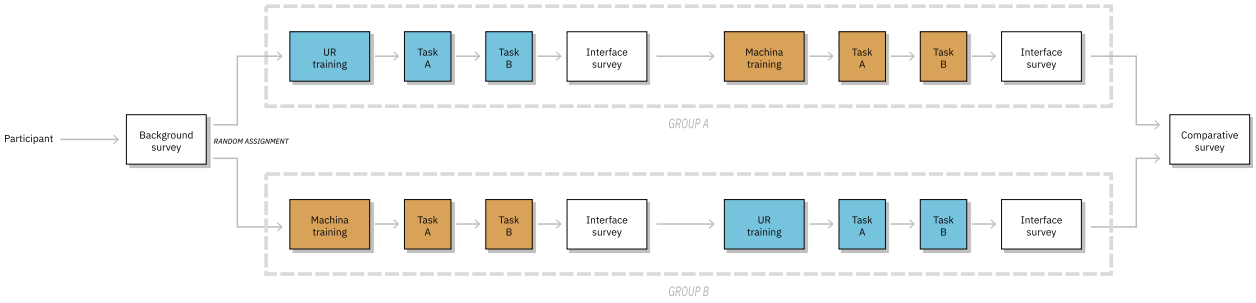


Figure 77 - Diagram of the crossover structure of the controlled user study.

In the study, a group of volunteers was recruited with no prior experience —or at least, so little that they self-reported as novice users. The participants were randomly divided among two controlled user groups A and B (Figure 77).

Users in *Group A* were asked to complete two rounds of training plus task implementation. On the first round, users were asked to complete the official training provided by the robot vendor, and use that knowledge and the default robot interface to implement two predetermined control tasks. Upon completion of the tasks, users concluded the round by filling in a feedback survey about their experience with the tools. On the second round, users were asked to complete a set of tutorials built around the Enactive Robotics model. After this training, users were asked to complete the same two control tasks, using the Machina Bridge application to program the robot (see §4.2.1), and to conclude the round with the same feedback survey about the new toolset. Upon completion of the two rounds of training and development, users would conclude the study by filling in a survey with their comparative opinion between the two platforms. Users in *Group B* followed the same structure for the study, but with the two rounds in reverse order: they would start with the Enactive Robotics

training on round one, to continue with the official vendor training on round two. This group differentiation is intended to avoid a possible comparative bias based on training order.

6.1.2 Users

Participants for the study were recruited from the student community at the Harvard University Graduate School of Design, and the Boston Tech Poetics group[33]. Calls for participation were posted on email distribution lists for both groups, and participants opted-in as volunteers. As compensation, users were invited to snacks during the testing, and the possibility to keep the outcomes of Task A for themselves; no economic retribution was offered.

Users were invited to participate on the study in a constant, controlled environment set. The test was conducted and monitored by the author. Some observations, as well as quantifiable metrics were logged by the author; users were made aware that some observations were being logged, but no information about their nature was provided until the test had concluded.

The study followed some of the general guidelines recommended by Gomoll in *Some Techniques for Observing Users* (1990), including:

- Participants were described the general purpose of the study prior to starting the test.
- Participants were notified that they were being observed, and that their actions were potentially being logged.
- Participants were explicitly made aware that the goal of the study was to evaluate the tools, not the users.
- Participants were introduced by the observer to the task to be completed at every stage of the test.
- Participants were allowed to ask questions before start and during testing.
- Participants were encouraged to "think" aloud.
- Participants were allowed to quit at any time during the test.
- Participants were given full context about the study and the observations upon conclusion.

In order to maintain consistency, users were informed of the context of the study and its rules by reading the same standard set of introductory information (see Appendix B: Controlled User Study Data, Introductory Information). Additionally, for each training round, users were given a "cheat sheet" with the commands/actions covered in the tutorials, to speed up the process and allow them to write notes on top (see Appendix B: Controlled User Study Data, Cheat Sheets).

[33] The Boston Tech Poetics is "a creative tech community in the Greater Boston Area". It is mainly comprised of individuals with interest in creative applications of technology, and their main activity is the organization of talks by authors discussing projects at the intersection of design, computers, science and art.

6.1.3 Training: UR Academy

The study was conducted using a Universal Robots (UR), model UR10, one of the leading brands on the market of collaborative robots. This is currently one of the leading brands in the market of collaborative robots. Their marketing material sells them as "Robotics Is Finally Within Your Reach," with adjectives like "simple," "easy to program," and "flexible" ("Robotics as it Should Be" 2015) at the forefront of their campaigns. As one of the most "user-friendly" (ibid.) option within industrial robotic arms, these robots were chosen as the best benchmark to test against for studies regarding accessibility and intuitiveness of use for wider audiences.

Figure 78 - Promotional material from Universal Robots, marketing them as "easy" and "user-friendly" ("Robotics as it Should Be" 2015).

Figure 79 - The training modules on the UR Academy platform.

UR robots ship with a custom programming interface called PolyScope ("PolyScope Manual" 2018), accessible through the device's teach pendant. The interface allows loading external programs written in the device's native URScript ("The URScript Programming Language" 2018). However, for novice users, the PolyScope interface features a suite of tools for manually jogging the robot, including a "freedrive" option to teach the robot program waypoints by physically dragging it with the hand. PolyScope also allows the live composition of robotic tasks, with the interface displaying a tree-like structure of the program, allowing the user to create sequences of commands and move back and forth between them. This feature is particularly relevant for the comparative goals of this study, as it is a non-explicit form of controlling-while-programming that aligns with the principles of the Enactive Robotics model (see §3.3).

Additionally, UR provides a free online training platform called "UR Academy" where users can "become a robot programmer in only 87 minutes" (Universal Robots 2018). Upon sign up, users can access nice video training modules (Figure 79), broken up into the following categories:

- First Look, Features and Terminology.
- How the Robot Works.
- Setting up a Tool.
- Creating a Program.
- Interaction with External Devices.
- Safety Settings.
- Feature Coordinates.
- Packaging.
- Program Flow.

Modules 1 to 4 introduce the trainee to the basics of robotic motion, how to create small tasks with the PolyScope interface and how to run them on the robot. Modules 5 to 9 are very specific to automation in manufacturing and/or advanced topics in robot programming, and fall outside of the scope of this research. For the purpose of this study, participants were required to complete only modules 1 to 4, reducing training time to approximately 48 minutes. Snapshots of these videos were archived for documentation purposes, and can be consulted through the links in Appendix B: Controlled User Study Data, Training Material.

Users were allowed to follow the training at their own pace, and optionally stop the videos at will to test their content on the physical device, available to them during training. This possibility was particularly important for the sake of this study, in how it relates to the action-based cognitive properties of the Enactive model.

By choosing a publicly available training set by a "leader in collaborative robotics" (Universal Robots 2015), and designed by the company to complement their "acclaimed" programming toolset, this study aims to provide a meaningful benchmark against one of the most state-of-the-art robot programming standards.

6.1.4 Training: Enactive Robotics

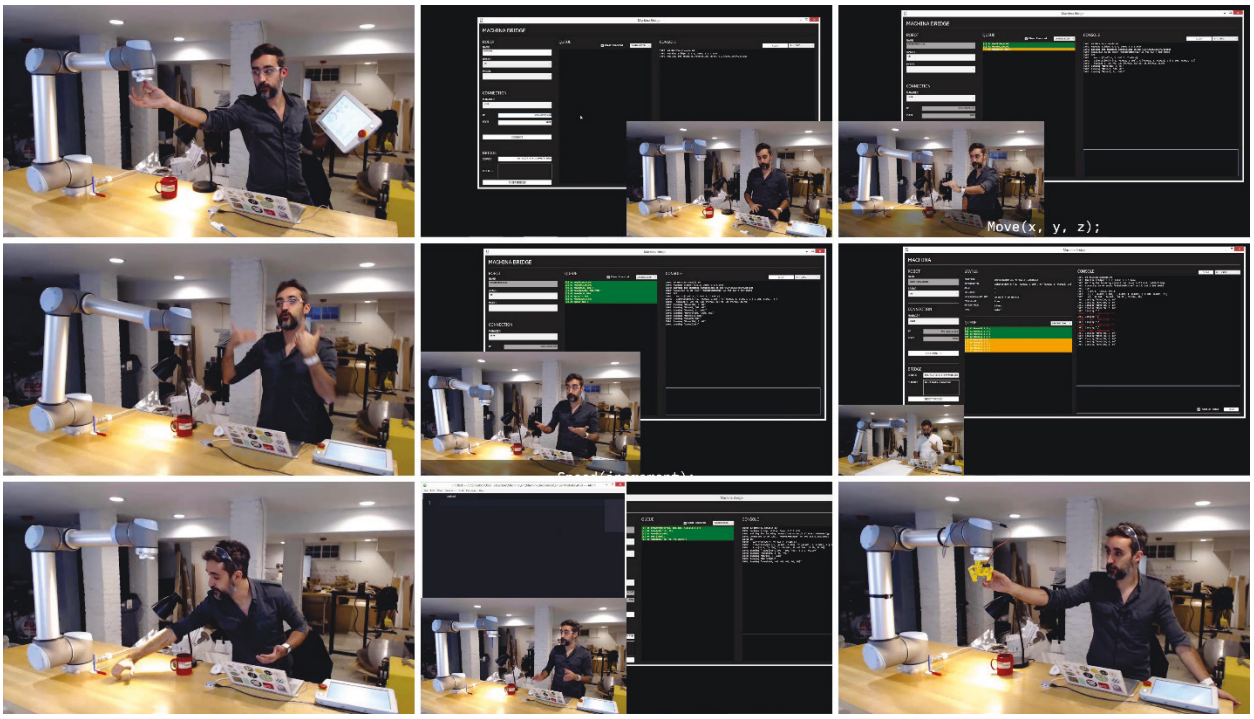


Figure 80 - The nine modules in the Enactive Robotics training.

The author's dissertation proposes Enactive Robotics as a new paradigm of robot control, built around the idea of concurrent actions as a helpful cognitive model towards assimilation of the medium for novice users, and as enabler of richer sets of interaction possibilities for advanced users. This is in opposition of models typically fostered by conventional robot programming tools, such as offline programming, which detaches the user from the medium and require abstract reasoning and training prior to any significant robot interaction.

In this study, users will be introduced to robotics following an enactive approach through a training set prepared by the author according to these principles. The training set focused specifically on the use of the Machina Bridge as the main interface for robot programming and control. The training set was designed to cover the basics of robot motion and control, introducing participants to simple motion and reorientation actions, as well as the logics of simple program composition. The video tutorials placed a strong emphasis on the physical robot as the medium to illustrate the concepts explained on each section; content was introduced by streaming actions to the robot and getting live feedback from their effect.

The Enactive Robotics training set featured the following sections (Figure 80):

- Using a Universal Robot
- Setting up Machina and connecting to the robot
- Basic motion

- Relative vs. absolute actions
- Speed and precision
- Queuing and action streaming
- Rotations
- Program Composition
- IOs: inputs and outputs

The training set was designed to be comparative in duration and scope of contents with the UR Academy one discussed in the previous section. The total duration of the playlist was approximately 38 minutes, and it can be consulted through the links in Appendix B: Controlled User Study Data, Training Material.

Similar to the UR training, users were allowed to follow the training at their own pace, and optionally stop the videos at will to test their content on the physical device, available to them during training. This possibility was particularly important for the sake of this study, in how it relates to the action-based cognitive properties of the Enactive model.

6.1.5 Task A

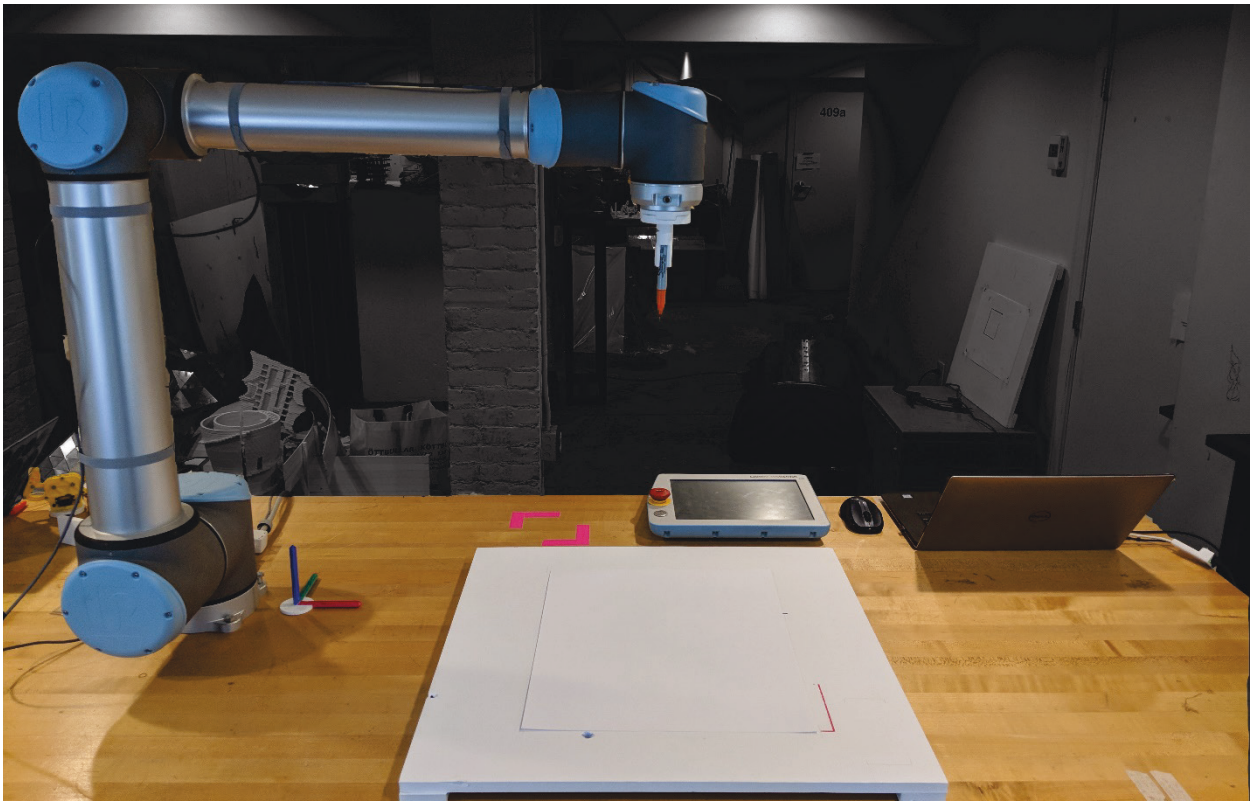


Figure 81 - Robot setup for Task A.

After each one of the training sets, either UR Academy or Enactive Robots, users will be asked to implement two robot tasks. The tasks will be the same, regardless of which training set users are coming from. The implementation of these tasks will be the main testing ground of this study.

In Task A, users were asked to trace a 100x100 mm square on a piece of paper, with a marker attached to the robot flange.

The marker holder, the paper and a soft platform on the table were provided for the task. All participants started with the robot at the same initial "home" position, and with the paper at the same relative location in front of the robot. Additionally, users were provided measuring tape for their optional use.

Successful implementation of the task typically involved:

- Spatial understanding of the setup.
- Estimation of the Z height of the paper base.
- Correct initial approach of the marker to the paper surface.
- Correct tracing of the square on the paper.
- Retracting of the marker from the paper surface.

The description of the task to the user was purposely open about the following aspects:

- The location of the square on the paper was not prescribed.
- The supporting foam core board was slightly unleveled, typically causing the marker to draw the square with stronger pressure on one corner, and potentially not touching the paper on the opposite. This posed an interesting challenge, as perfectly traced squares usually required more complex programs that introduced non-planar trajectories.

Users who inquired about any of the above aspects were given freedom to choose how to respond to them.

Users were asked to self-report when they considered they had accomplished the task, with the task considered complete upon this report. Time between task declaration and user self-reported completion was tracked and internally logged by the observer; to avoid biases, users were not notified about this measure.

This task was designed as a simple, introductory task, with the capacity to be implemented in a short period of time and resulting in a tangible drawn object. The task could potentially be implemented with just six motion commands/actions, and required no spatial reorientation of the marker.

6.1.6 Task B

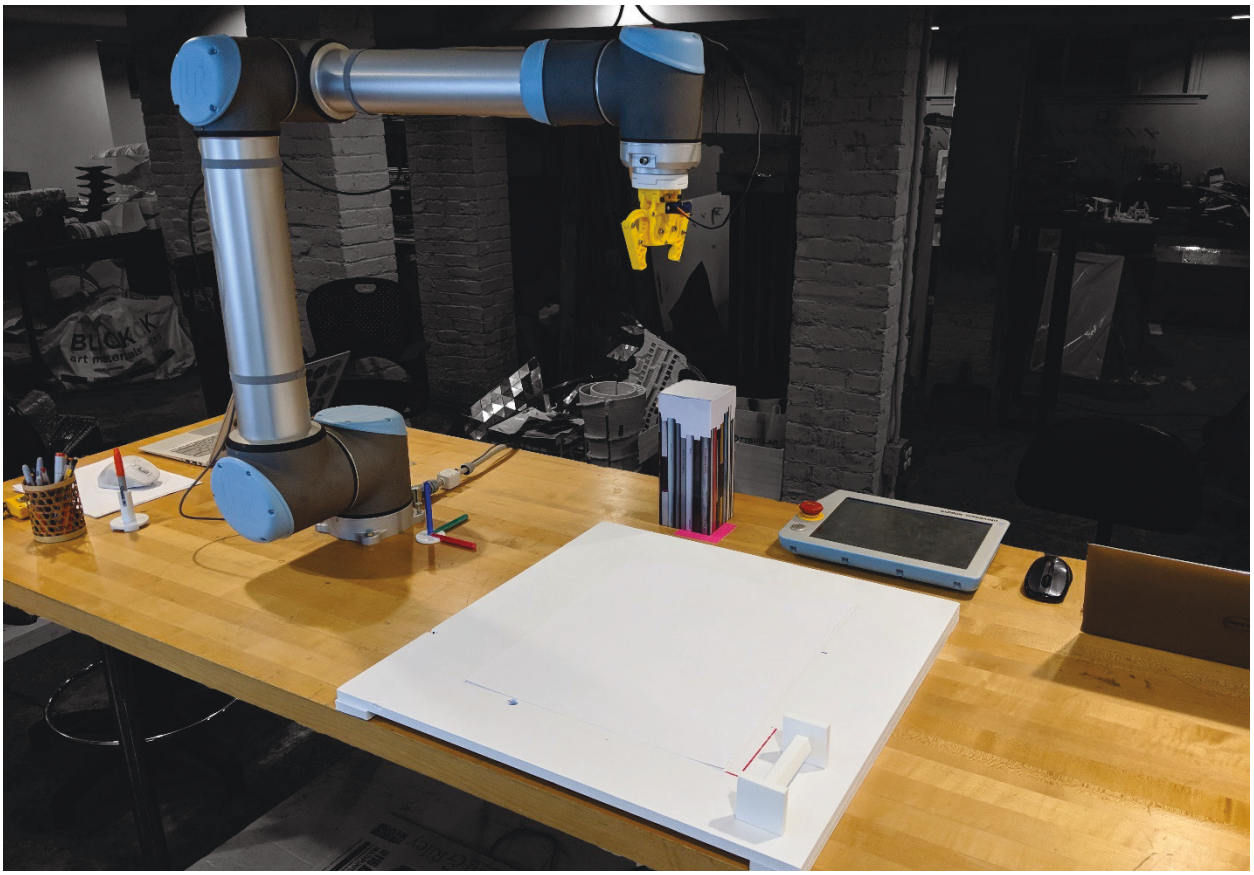


Figure 82 - Robot setup for Task B.

In Task B, users were asked to pick up an object from the foam core platform with the grippers attached to the robot flange, and drop it on a specified elevated platform.

The gripper, object and drop-off platform were provided for the task. All participants started with the robot at the same initial "home" position, and with the object and drop-off platform at the same initial locations. Additionally, users were provided measuring tape for their optional use.

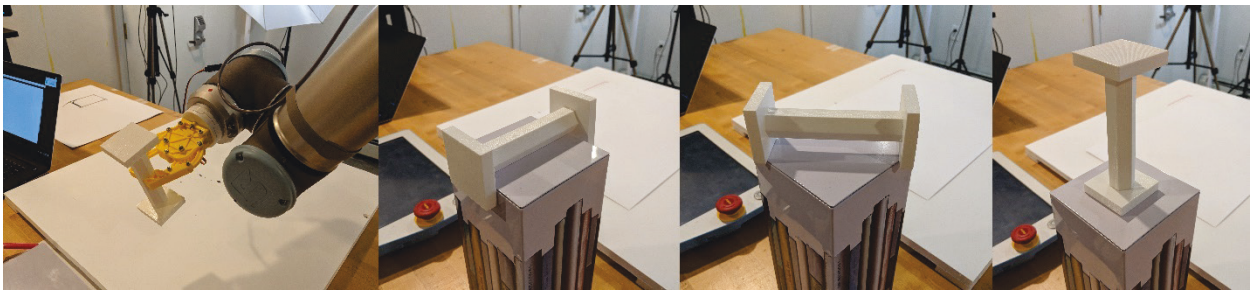


Figure 83 - Custom gripper and pick-up object (left), with possible orientations for drop off (rest).

The gripper was an assembly of custom 3D printed parts operated by a servo motor and controlled via an Arduino board connected to the robot's IO system (Figure 83). The gripper was controlled via a digital output, accepting only open and closed states; users were notified of which pin controlled the gripper's servo.

The pick-up object was a custom 3D printed body comprised of a main hexagonal shaft with two larger square stands on the ends. The section of the shaft was designed to fit optimally with the shape of the gripper tips, with the stands making the shaft sit over ground level on a horizontal position. This design facilitated the process of picking up the object. Furthermore, the length of the shaft was designed to slightly exceed the length of the side of the drop-off platform. This constrained the object orientation on drop-off to three possibilities: overflowing the platform perpendicularly, standing on the platform in a horizontal diagonal, or standing on the platform vertically (Figure 83).

Successful implementation of the task typically involved:

- Spatial understanding of the setup.
- Estimation of the Z height of the object.
- Rotation of the gripper to align with the shaft.
- Correct initial approach of the gripper to the object's shaft.
- Waiting to reach position before gripping.
- Gripping the object.
- Waiting to grip the object before retracting.
- Retracting vertically.
- Approach the drop-off platform.
- Optional rotation of the gripper to drop-off orientation.
- Drop off of the object.

The description of the task to the user was purposely open about the following aspects:

- The trajectory of the robot between pick up and drop off.
- The spatial orientation of the object upon drop off.

Users who inquired about any of the above aspects were given freedom to choose how to respond to them.

Users were asked to self-report when they considered they had accomplished the task, with the task considered complete upon this report. Time between task declaration and user self-reported completion was tracked and internally logged by the observer; to avoid biases, users were not notified about this measure.

The task was designed to demand an increased level of calibration, planning and a wider range of commands/actions for successful program execution. Performing the spatial rotations required to pick up the

object often required anticipation and trial-error. Additionally, there was the need to account for the lag time between turning on the gripper pin and the gripper fully closing on the object; this often required the introduction of "wait" commands in the program to avoid false grips. Furthermore, the three possible alternatives on how to drop the object required the user to make decisions about how to invest their programming efforts: in fine-tuning a precise position for perpendicular drop off, or in adding additional spatial rotations to the program to drop it otherwise.

6.1.7 Surveys

User were surveyed four times during the course of the test.

The first survey "Background Information" was a general background survey, in order to assess the demographics, skill level and robotic experience of the user. The objective of this survey is to assess that the user's background is related to creative fields such as design, architecture or art, to evaluate their skillset and to identify them as novice robotic users. Sample questions in this survey include:

- Age/gender
- Current occupation.
- Experience level on different technical platforms.
- Have you ever used an industrial robot?
- Why did you volunteer to participate?

Users finished each one of the two rounds of training and robot implementation by completing an "Interface" survey. This questionnaire was designed to qualitatively assess the user's experience learning with and trying to use the main robot programming system to implement a robotic task. Sample questions in this survey include:

- Did you find the training tutorials useful to understand how to use the robot?
- Do you think testing things out on the physical robot while in-training helped you understand it better?
- How did you like programming the robot with this system (PolyScope or Machina)?
- Did you find it easy to use? And intuitive?
- How easy was to implement each task?
- What do you think are the best and worst features on the interface?

Finally, the test would conclude with a final "Comparative" survey. Once the user has learned both systems, and developed the same tasks with each one of them, this survey inquires about their experience comparatively between systems. Sample questions in this survey include:

- Which system did you find easier to learn? And intuitive? Which one allowed better iteration/experimentation?
- Which system made implementation easier for each task?
- Which system would you choose if you were to use the robot again?

A complete account of all the survey's questions and answers for each participant can be found on Appendix B: Controlled User Study Data, Survey Data.

6.2 Results

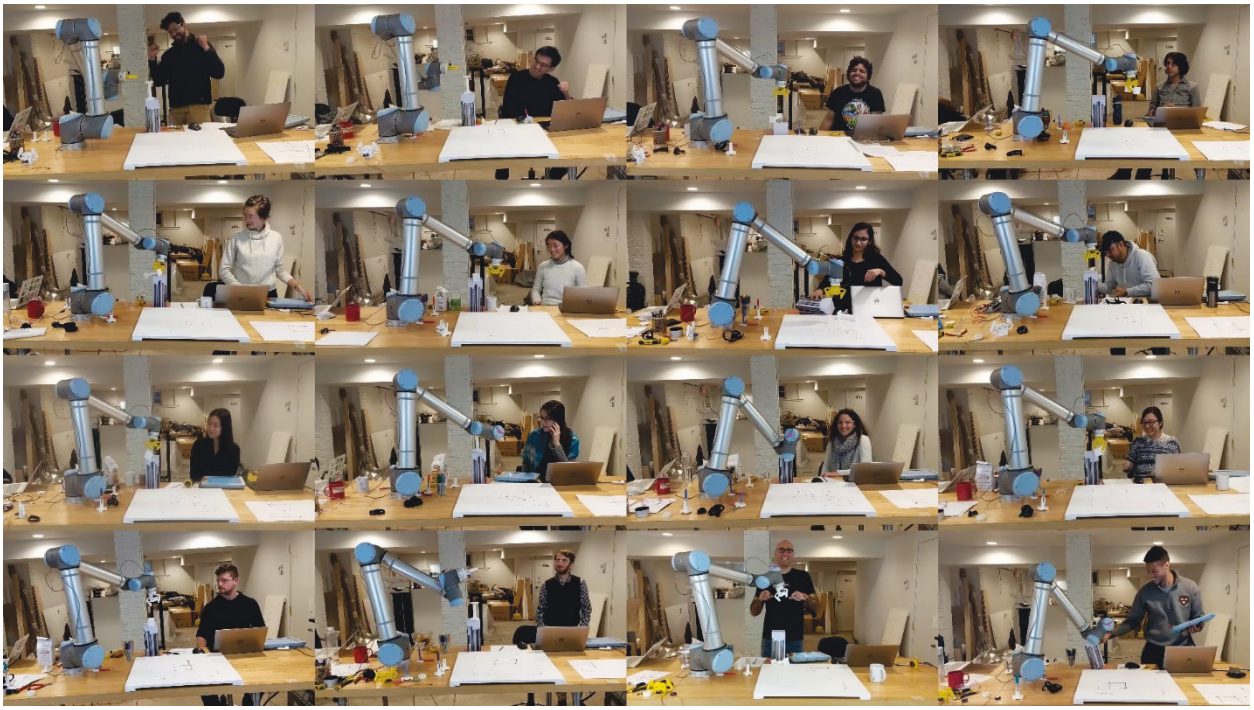


Figure 84 - Participants in the controlled user study.

20 volunteers participated in the study, which was conducted between November 15th and December 1st, 2018. Full tests were completed on average in around three hours, with some participants requiring up to four, and some finishing in almost two. A total of 22 participants signed up for the study, but the first two users were unable to complete the tests due to technical problems with the setup, and were removed from the final sample set.

6.2.1 Summary of Findings

The results of this study show evidence to support the thesis claimed in this dissertation. In particular, the results back the assertion that the cognitive model fostered by the Enactive Robotics model makes learning and using robots easier, faster and more intuitive, while at the same time, providing a more fluid experience that fosters experimentation and iteration.

Overall, users found Machina easier and more intuitive to learn than PolyScope. The group was overwhelmingly more engaged when learning about robots through the enactive training, and most of the users manifested how interacting with the robot while in-training was of great aid in the learning process, helping them better understand how to control the machine.

The main feedback gathered around the Machina Bridge was how it provided an easy and intuitive platform to understand and interact with the robot. Users were particularly excited about the immediacy of typing an action on the command-line interface and immediately seeing its execution on the machine; users reported how this featured allowed them a rather fluid interaction with the machine, and the possibility to try things out fast and learn through trial and error. On the contrary, users found the `Action/To()` syntax problematic. While they understood the conceptual difference between relative and absolute actions, they were often confused about the similarity of their instruction form, resulting in misspelled requests that required emergency stopping of the program. Similarly, users expressed elevated levels of uncertainty about the results of their actions, and wished for the system to prevent them from executing "erroneous" requests.

Most users reported the "freedrive" option on the UR as the system's greatest asset. This feature allowed them fast and immediate implementation of complex spatial transformations, without needing to generate their numerical representation. Additionally, this feature made them feel more confident about their programs, since each waypoint had been previously defined in physical space by manually moving the robot to that location. However, most users also agreed that the UI provided by PolyScope was deficient, offering a confusing tree structure for the program, requiring tedious jumps between menus and making it rather difficult to edit the parameters of previous commands. Furthermore, some users considered that the interface didn't allow them to enter motion through "precise" coordinates, resulting in implementations of Task A were the side of the square was manually "estimated" with the help of rulers or tape measure.

The main critique to both systems was the lack of a clear representation of the coordinate system of the robot. Users were often confused about the robot's coordinate axis and their orientation, and wished for a clearer representation as part of the system. They also manifested the desire for an enhanced feedback system, where commands/actions could be pre-visualized digitally before execution.

Upon comparative survey, most users manifested preference for the Machina framework in all aspects inquired. Users found themselves significantly more comfortable programming Task A with Machina, and the task took on average nearly half the time to develop with this system than with PolyScope. Regarding Task B, the

majority still leaned towards the Machina environment, but results were slightly more evened out. Completion times for Task B were comparable among both systems, with the "freedrive" option on UR proving significantly advantageous to implement the spatial rotations required in this case.

The majority of the participants perceived the target user base of the UR/PolyScope system to be found among the engineering and manufacturing fields, whereas Machina was deemed a much more suitable platform for creative individuals, tinkerers and users with some programming experience. It is noteworthy however how most participants identified both systems as adequate for non-technically savvy individuals in each of those fields, and as good introductory platforms.

Finally, most users manifested how, if ever in need to use robots again, they would choose Machina as their preferred tool.

6.2.2 Background Survey

The final sample set was comprised of 20 participants, with an equal distribution of users (50%) assigned to each group A and B.

The group demographic showed a distribution of 20% users between the ages of 22-25, 50% between 26-30, and 30% between 31-40, with gender parity (50%) across all participants. 60% of participants identified themselves as students as their primary occupation, while the rest (40%) self-reported as mainly involved in professional activities.

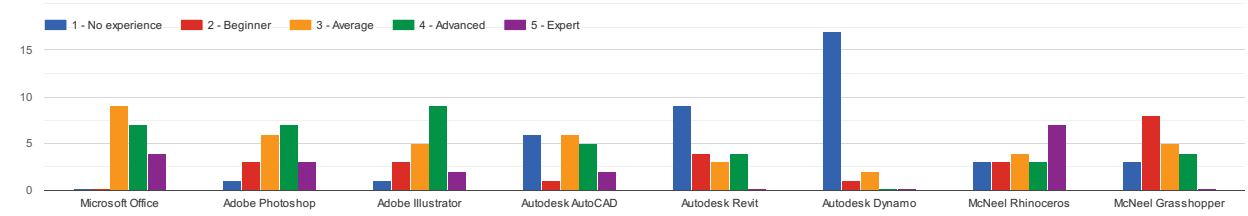


Figure 85 - Responses to the question "Please rate your experience level with these software packages" in the "Background" survey.

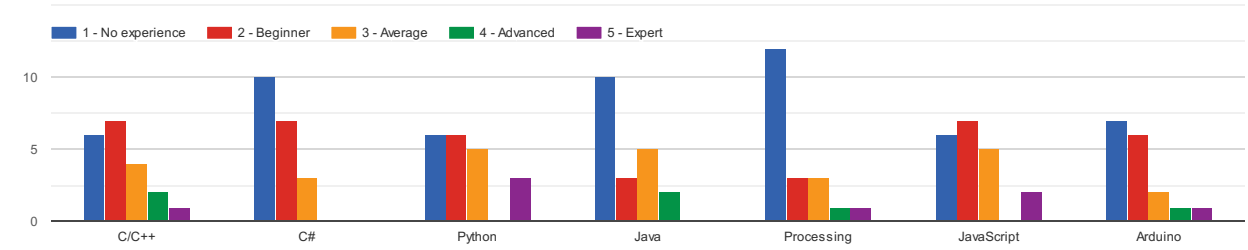


Figure 86 - Responses to the question "Please rate your experience with these programming languages" in the "Background" survey.

The group showed an above average distribution of expertise in software packages commonly associated to design professions (Figure 85), like Adobe Photoshop and Illustrator (3.40 avg.), as well as software platforms commonly associated with 3D modeling and CAD/CAM processes, like Autodesk AutoCAD (2.80 avg.), McNeel Rhinoceros (3.40 avg.) and Grasshopper (2.50 avg.). 85% of the users reported having 3D printed an object in the past. Most of the users also reported as having little or no experience with most programming languages (Figure 86), with Python being the most popular among participants (2.40 avg.) and C# the least (1.65 avg.).

85% of the participants reported having never used an industrial robotic arm in the past. Among the rest (15%), all of them (3) reported having used ABB robots, one (1) single user had used a KUKA robot, and no participant (0) had ever used a Universal Robot. When inquired about the tools they had used to drive robots, all users (3) reported having used RobotStudio, while additionally, some participants had also used HAL, TacoABB, Machina and SuperMatterTools (1 ea.).

When inquired about the reasons to volunteer for the study, most participants manifested curiosity about learning robotics, using a robot arm, and how they could apply it to their own fields.

The survey shows that the sample user base matches the community of users that the Enactive Robotics and this dissertation target: individuals in the creative fields, familiar with CAD/CAM tools and with little to no experience in computer programming.

6.2.3 Interface Survey

In this section, an annotated breakdown of the answers reported on the "Interfaces" survey is presented.

In line with the crossover methodology of this study (see §6.1.1), metrics have been split into two control groups. Data reported under the label "ROUND 1" has been calculated taking into account only the responses corresponding to the first round of training plus task implementation (10 answers per system). This data reflects the opinions of users after their first experience with robot programming, and is the foundation of this study. Data reported under the label "ROUND 2" or "TOTAL" has been calculated aggregating the responses of each participant for both the first and second rounds of training plus task implementation (20 answers per system). This data shows the opinion of all users for both platforms, and may reflect biases that the users carry over from the first tested system into the second.

Each question includes the author's interpretation of the meaning of the data presented, as well as data summary tables and selections of literal responses where applicable.

Did you find the training tutorials useful to understand how to use the robot?

	MACHINA	POLYSCOPE
ROUND 1	4.50	4.50
TOTAL	4.50	3.85

Table 4 - Average scores for the question "Did you find the training tutorials useful to understand how to use the robot?", from 1 = "Not at all" to 5 = "Very useful."

Users found both tutorial sets equally useful to learn the basics of robot programming. This suggests the preparation of the Machina Training set was a good fit to match the content of the UR Academy training. However, once undertaken a second round of training, rates of usefulness decayed on the PolyScope side, suggesting the training set added less actuable content.

If 2 or under, why not?

Selected responses (PolyScope):

> "Couldn't really use robot arm during process. Abstract."

How engaged were you during the training?

	MACHINA	POLYSCOPE
ROUND 1	4.80	3.60
TOTAL	4.35	3.05

Table 5 - Average scores for the question "How engaged were you during the training?", from 1 = "It was sooo boring" to 5 = "It was fun!"

Users manifested much higher rates of engagement when learning with the Machina platform, suggesting that the Enactive model is better suited as a cognitive tool. Rates of engagement declined similarly after the second round of training.

Do you think testing things out on the physical robot while in-training helped you understand it better?

	MACHINA	POLYSCOPE
ROUND 1	4.90	3.40
TOTAL	4.65	3.90

Table 6 - Average scores for the question "Do you think testing things out on the physical robot while in-training helped you understand it better?", from 1 = "Not really" to 5 = "Super helpful."

Users clearly perceived that "enacting" while learning contributed very positively to their understanding of the system. The increase on the PolyScope training might be a carryover opinion from having used the Machina framework first.

How did you like programming the robot with this system (PolyScope or Machina)?

	MACHINA	POLYSCOPE
ROUND 1	4.00	4.00
TOTAL	4.10	3.50

Table 7 - Average scores for the question "How did you like programming the robot with this system (PolyScope or Machina)?", from 1 = "Not at all" to 5 = "Very much."

Users seemed to enjoy both systems equally on their first experience with the robot. However, rates declined on second round for PolyScope, suggesting the initial excitement wearing off and Machina being perceived more positively.

If 2 or under, why not?

Selected responses (Machina):

> "Small things like 1 != true, working on a multi-line program and having it erased if I went up too far (and went further up in history), and other firmware related issues. I also wished that it showed a simulation before running so I can check what would happen."
>
> "not very intuitive but still fun"

Selected responses (PolyScope):

> "The UI felt clunky and unintuitive to me. For example, I didn't understand the nesting structure of the program tree."
>
> "A bit passive and not working with the machine itself."

Did you find it easy to use?

	MACHINA	POLYSCOPE
ROUND 1	4.20	3.40
TOTAL	3.95	3.40

Table 8 - Average scores for the question "Did you find it easy to use?", from 1 = "Very difficult" to 5 = "Very easy."

Users found Machina generally easier to use.

Did you find it intuitive?

	MACHINA	POLYSCOPE
ROUND 1	3.90	3.20
TOTAL	3.95	3.15

Table 9 - Average scores for the question "Did you find it intuitive?", from 1 = "Not at all" to 5 = "Very intuitive."

Users found Machina generally more intuitive.

Did you ever feel frustrated?

	MACHINA	POLYSCOPE
ROUND 1	Yes: 3, No: 5, Other: 2	Yes: 6, No: 4, Other: 0
TOTAL	Yes: 8, No: 9, Other: 3	Yes: 13, No: 7, Other: 0

Table 10 - Responses the question "Did you ever feel frustrated?", with options "Yes," "No" or "Other."

On both rounds, PolyScope yielded a higher level of frustration. Most of the reasons were related to the UI design, and the tree organization of the program sequence. On the Machina side, frustration was generated by confusion around the `Action/To()` action syntax, and the difficulty to figure out rotations in space. Participants on both systems manifested difficulty in understanding the coordinate system of the robot.

If "Yes", why?

Selected responses (Machina):

> "Action() and ActionTo() were a bit confusing because they are too similar in words compared to the big difference in its consequence. Also it was hard to see the x and y axes in real life."
>
> "I still had trouble understanding how the coordinates work. My mind was mismapping them to the machine."
>
> "The rotations part confused me."

Selected responses (PolyScope):

> "Sometimes I didn't understand the nesting structure of the program tree. Or, I had to mentally model how I thought the robot would perform. Sometimes my mental model was wrong, and this felt frustrating."
>
> "clunky interface, not easy to add instructions in the right place"

> "I felt frustrated at how many menus I had to click through to write a simple command when I already knew what I wanted the robot to do."

Did you feel the system provided you with a fluid programming experience?

	MACHINA	POLYSCOPE
ROUND 1	4.10	3.40
TOTAL	3.95	3.15

Table 11 - Average scores for the question "Did you feel the system provided you with a fluid programming experience?", from 1 = "Not fluid at all" to 5 = "Very fluid."

Users believed Machina provided a more fluid programming experience. They manifested PolyScope required a lot of effort, and there was significant confusion around the structure of the UI. On the other hand, users reported problems related to code development on the Machina Bridge side, such as the lack of code completion, or the convenience of including additional UI-based options.

If 2 or less, why?

Selected responses (Machina):

> "This feels like it's in Arduino land of robotics. I want to use something that is tactile and mimics the physical reality of the robot moving. The layer of abstraction between the LOGO-esque programming environment and the robot arm made the experience more effortful and less fluid."

> "If I didn't know coding, I probably would have been looking for more GUI interface/assistance."

Selected responses (PolyScope):

> "It did not feel fluid to me for several reasons. One was that there was a great deal of clicking between tabs. Another was having to understand the nesting structure of the tree."

> "Getting anything done took a lot more effort; I had to move the robot to each waypoint before I try anything out. That said, it was valuable for programming the robot because I ended up with a complete program, whereas with Machina, I tended to run commands in the interpreter to live-program the machine rather than create a reusable program."

Task A - Completion time

	MACHINA	POLYSCOPE
ROUND 1	9:31	18:50
TOTAL	9:22	14:49

Table 12 - Average completion times for Task A in minutes. Task completion was determined by users self-reporting. To avoid bias, users were not aware task time was being tracked.

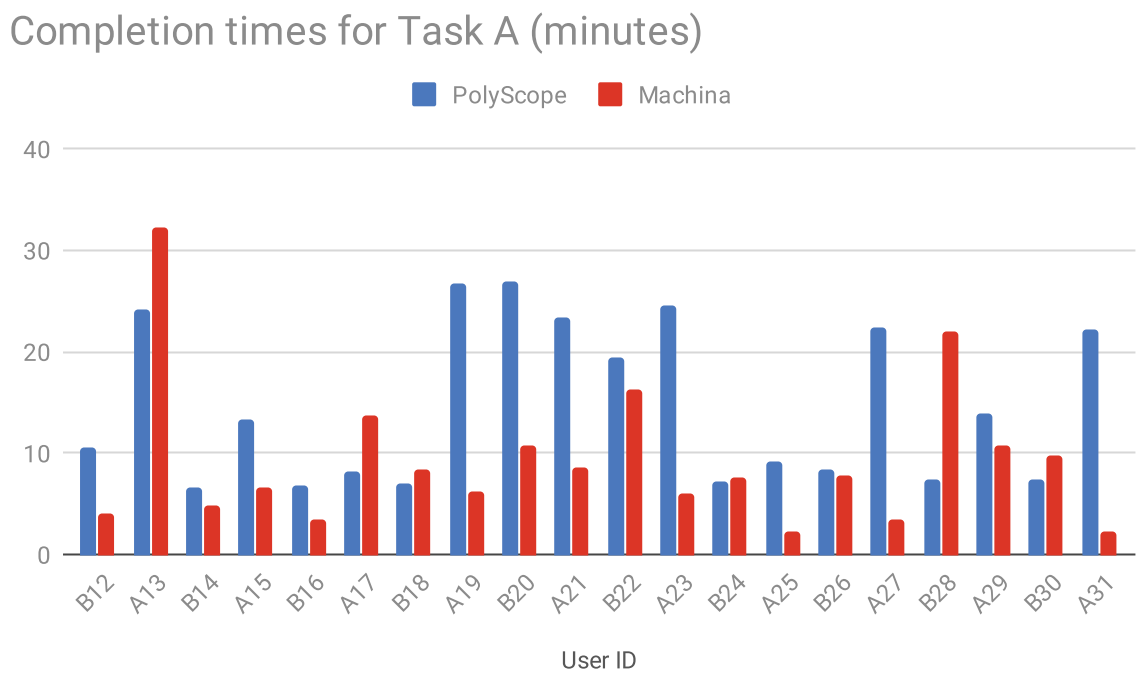


Figure 87 - Completion times in minutes for Task A per user.

Users implemented "Task A: Draw a Square on Paper" significantly faster with Machina. On second round, prior robot knowledge contributed to speed up task implementation.

Task A - Was it easy to implement?

	MACHINA	POLYSCOPE
ROUND 1	4.50	3.40
TOTAL	4.35	3.30

Table 13 - Average scores for the question "Task A - Was it easy to implement?", from 1 = "Very difficult" to 5 = "Very easy."*

Users found Machina a much easier tool to draw a squared shape on paper.

Task A - What was the most difficult part to figure out?

On the Machina side, users reported problems calibrating the marker height according to the paper and finding the coordinates of the first point of the drawing. Furthermore, users found it harder to account for the sloping of the board with a Machina program than with the PolyScope interface.

On the PolyScope side, users manifested trouble switching from "freedrive" to inputting exact coordinates on the system. This resulted in several users estimating the dimensions of the square with a ruler, rather than implementing them numerically via PolyScope. Furthermore, users felt editing previous parts of the program was challenging.

Users manifested difficulty on both platforms in adapting their mindsets from imperial to metric system.

Selected responses (Machina):

> "Finding absolute coordinates felt challenging; I didn't have a good sense for where I was on the canvas at the start (some graphical feedback may be helpful). Once the first point was placed, the system for programming the rest was very straightforward."

Selected responses (PolyScope):

> "Height for the pen / sharpie. Had to redo all waypoints when replacing the sharpie bc of a slightly different height. Wasn't clear how to edit just one direction."

> "I found that it was difficult to figure out where to place the waypoints. I ended up cutting corners and making an approximate square, by just manually positioning waypoints close to 100mm away."

Task B - Completion time

	MACHINA	**POLYSCOPE**
ROUND 1	17:05	21:51
TOTAL	18:34	18:46

Table 14 - Average completion times for Task B in minutes. Task completion was determined by users self-reporting. To avoid bias, users were not aware task time was being tracked.

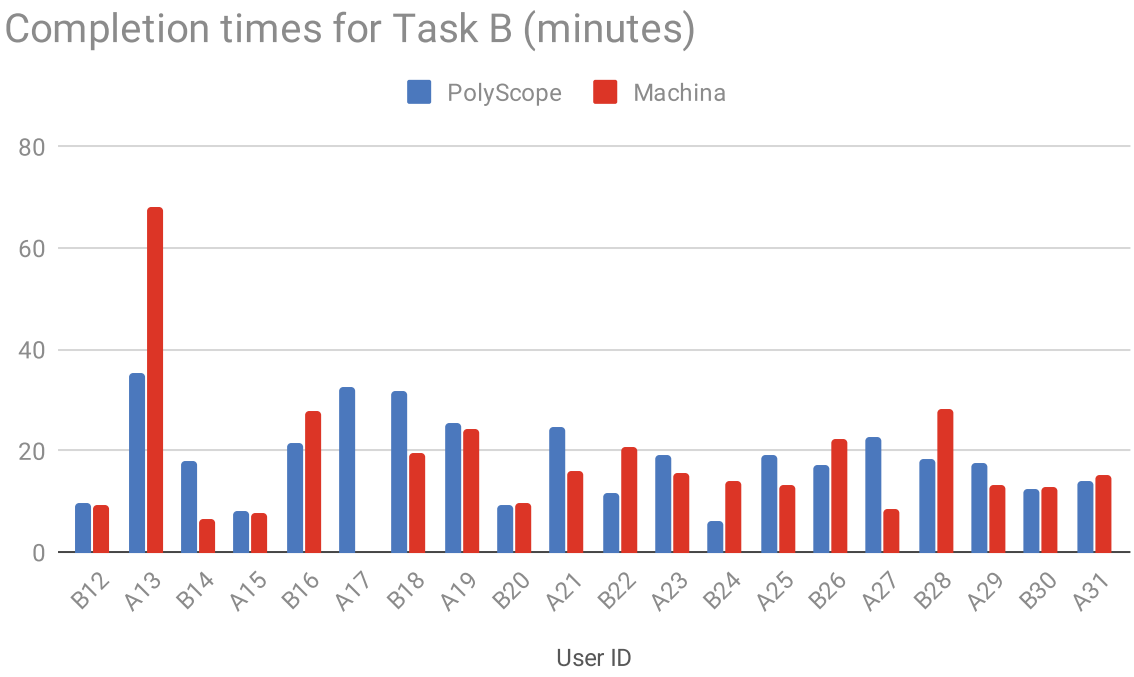


Figure 88 - Completion times in minutes for Task B per user.

There is no meaningful difference on implementation times between the two systems for "Task B: Pick and Place an Object." The fluidity of the Machina system was matched by the fluidity of the "freedrive" mode to implement complex spatial transformations.

Task B - Was it easy to implement?

	MACHINA	POLYSCOPE
ROUND 1	3.90	3.70
TOTAL	4.00	3.40

Table 15 - Average scores for the question "Task B - Was it easy to implement?", from 1 = "Very difficult" to 5 = "Very easy."*

In line with the metrics for completion time, users were equally divided about task difficulty on first round. However, users who had previously used Machina find the task more difficult to implement in PolyScope on second round.

Task B - What was the most difficult part to figure out?

On the Machina side, users found it hard to determine the location of significant objects on the task in relation to the robot's coordinate system. For PolyScope, users identify gripping the object as the most challenging part.

Selected responses (Machina):

> "There were a lot of components involved. It was difficult coordinating the z axis low enough to pick up the object. As well as deciphering which way the rotate command line was working on the wrists."

> "The coordinates of each object; there's so much uncertainty that I was afraid to program absolute coordinates for the task and chose to solve it iteratively / through live programming instead."

Please name the features you liked the MOST about the way you programmed the robot with this system.

On Machina, participants appreciated the precision the system gave them, along with the immediacy of typing an action and executing it immediately. On PolyScope, most participants expressed affinity for the capacity to move the robot around manually and determine the waypoints through "freedrive", very especially after having used Machina before. They also manifested how they felt "safer" this way since, as they had moved the robot physically first, they had a greater degree of confidence about the correct location of the waypoints in the program.

Selected responses (Machina):

> "The system itself is very easy and intuitive to use without any further in depth training for basic tasks."

> "I liked that I could give the robot commands at a console and see results live, rather than writing an entire program first."

> "I could see this system being much more friendly and compatible to work with for people with a background in programming, especially in coordinate-based systems like Processing etc. I enjoying being able to write a single line of code for control much more than having to click through multiple menu options on the other interface."

> "Really liked how quickly I could iterate. Very easy to have an idea of what I want to do in my head and execute."

Selected responses (PolyScope):

> "Flexible waypoint setting (either with freedrive or the GUI control system), ability to test a single point at a time while creating a sequence, generally very straightforward and feels risk-proof"

> "For Setting waypoints, I liked being able to manually manipulate it with my hands, then fine tune it with the controls. I felt like I was able to get what I wanted done very quickly"

> "It made me move the robot to each position first, so I was less afraid of breaking tools on the TCP and generally knew where the robot arm would end up."

Please name the features you liked the LEAST, or those you consider are MISSING.

Aside technical issues in Machina, most users expressed how the environment provided them with no feedback on the coordinate system of the robot before executing an action. Furthermore, users express the desire to for

Machina to prevent them from requesting actions that resulted in "bad" movements, such as out of range locations, self-collisions or punching the marker through the table. The main issue among PolyScope users was the perceived incapacity[34] to describe waypoints numerically rather than with freedrive. Responses also complaint about UI design, the program tree structure, and the touch screen not working properly.

Selected responses (Machina):

> "The rotation part was tricky to understand in theory because I am not used to using 3D tools."
>
> "Could use some visual representation of X&Y coordinates relative to where the arm was."
>
> "Understanding limits...how far the robot can go / can't go. A visual for more complex movements (rotations etc.) on screen could be helpful."

Selected responses (PolyScope):

> "The programming interface was cumbersome and hard to use on a touch tablet. I would have preferred code that I could manipulate and see all of the waypoints written out, since they are all relative to each other."
>
> "Miss directly interfacing with the programming - seemed very abstract what I was doing on the pad."
>
> "Quickly prototyping a task was exceedingly tedious and difficult, and took a long time to just get started. I still don't feel like I could generate toolpaths efficiently with PolyScope."

Please note any major problems you encountered while programming.

Machina users found the `Action/To` syntax for relative/absolute motion very confusing. Furthermore, they requested a way to stop program execution without having to e-Stop the robot. PolyScope users reported similar problems as in other questions: difficulty to understand the UI, change parameters on a program or define locations precisely.

Selected responses (Machina):

> "Action() and ActionTo() were confusing."
>
> "Many times I had to stop the robot after sending a job for safefty reasons"
>
> "I had to stop the machine multiple times because I used MoveTo instead of Move"

Selected responses (PolyScope):

> "The only programming issues I experienced were with understanding how to move the robot with coordinates. That was not clear in the tutorial because it only has you drag the robot physically into position."

[34] The PolyScope interface does indeed provide an option to input numerical information to change waypoints. However, it is not covered on the UR Academy tutorials, and it could be argued that the feature is not very obvious UI-wise.

"Going back into each waypoint or node to fix or change something."

"Seemed very abstract."

Please note any suggestions you may have to make the experience better.

In general, Machina users advocated for a higher degree of feedback from the system, such as pre-visualization of actions before they execute, clearer coordinate system and enhanced safety checks. PolyScope users pointed at the UI as a point for improvement, demanding less back and forth between tabs and better options to input numerical coordinates. Users on both platforms requested a better way of visualizing the coordinate system of the robot in relation to the real world.

Selected responses (Machina):

"Some sort of way to visualize X&Y besides numbers."

"It might be valuable to have Machina throw runtime errors for invalidly formed commands, rather than executing all valid commands in the queue and skipping invalid commands."

"This is a stretch goal, but it could be nice to have some sort of projector that projects the coordinates onto the surface, so you as the programmer don't have to have a model of where things are in physical space."

Selected responses (PolyScope):

"UI redesign. More (visual) feedback to reinforce the effects of the program on the movement/action of the robot"

"Like I said earlier, having something that very clearly marks what the x, y, and z axis map to in the real world is the thing I would find most useful"

"The interface of the polyscope could be improved in order to create a more user-friendly experience - from jumping between different tabs to create one command, to the touch screen accuracy."

Who do you think is the target audience of this robot programming system?

In general, participants considered that Machina is better suited for users with 3D spatial knowledge and some programming experience. However, participants also pointed at kids, tinkerers, creative individuals, designers and people without engineering backgrounds as potential users of the platform. For PolyScope, participants perceived that it is a better tool for people in the engineering/manufacturing business, with no programming experience and for programming repetitive tasks. However, they note how these audiences would not need to be tech-savvy, and how PolyScope can also be a good introductory tool.

Selected responses (Machina):

"Depending on the scale of the robotic arm, the audience could be anyone from architecture/design students to on-site construction contractors."

> "Probably people with at least a moderate programming background but are not professional engineers; maybe creative technologists and designers for exploring human-robot collaboration."
>
> "I see this as appealing to people who use Arduinos, or are just getting into programming."

Selected responses (PolyScope):

> "The target audience of the training software seems to be more industrial users (factory processes, etc) but the demo tasks feel more creative - it seems there could be a wide range in audience."
>
> "I think this is perfect for students or anyone first getting into robotic systems. I have no experience with any sort of programming other than a CS class way back in high school, so this was relatively easy to follow with."
>
> "People who do not know how to program. Perhaps people who want to automate their factories and want their factory workers to use this."

6.2.4 Comparative Survey

In this section, a breakdown of the answers reported on the "Comparative" survey is presented.

This survey was presented to users after completion of both rounds of training and task implementation, at the time when users had a holistic perspective on the basic robot programming models provided by the two systems tested. The survey focuses mainly on the users' comparative opinions across both experiences.

Which model did you find easier to learn?

75% of users (15) found Machina easier to learn.

Which system helped you better understand how the robot works?

60% of users (12) found Machina helped them better understand how the robot works.

Which system did you find more intuitive to use?

95% of users (19) found Machina more intuitive to use.

Which system made implementing the tasks easier? Task A

85% of users (17) found it easier to implement Task A on Machina.

Which system made implementing the tasks easier? Task B

65% of users (13) found it easier to implement Task B on Machina.

Which system allowed you better experimentation/iteration?

95% of users (19) found Machina allowed them better experimentation and iteration.

What did you learn from the first system, that helped you understand the second one better?

Responses to this question were fairly varied, and are considered inconclusive.

Which system would you choose if you were to use a robot again?

80% of users (16) would choose Machina again, over a 10% (2) who would prefer PolyScope. The remaining 10% (2) would opt for using a combination of both:

> "Honestly I would use both. I would like more time in understand the world in which the robot lives in the coordinates with Polyscope, however when getting into programming after awhile, I'd much prefer Machina. I find it more intuitive and precise."

> "I could see myself using PolyScope for actions that require more rough positioning like grabbing objects, I would use Machina for more precise actions like drawing the square."

6.3 Conclusions

In this chapter, the results of a controlled user study were presented. The study was designed to test the capacity of the Enactive Robotics model —using a concrete implementation thereof— to systematically provide easier and more intuitive access to robot programming and control for novel users.

A group of 20 volunteers participated in the study. The group was mainly composed of young designers and technology-oriented creative individuals, with no prior experience in using industrial robotic arms. Each participant was asked to complete two rounds of robotic training, each with a different robot programming system, and implement the same two tasks with each system afterwards. The users were surveyed about their experience at each stage of the test.

The systems compared in the study were the Machina Bridge, chosen as a sample technical implementation of the Enactive model, and PolyScope, Universal Robot's native programming platform, commonly published as one of the easiest, most user-friendly options in the market. After completing short training courses for each system, participants were asked to use that system to program the robot to perform two simple tasks: draw a square on paper with a marker, and pick an object from the table and drop it on a platform. The implementation of the tasks served as the comparative benchmark between systems; participants were surveyed after completion of the tasks with each system, and comparatively at the end of the test. To avoid

comparative biases, the experiment was conducted as a crossover study, with the sample population divided into two groups and each half conducting the tests with the systems in different order.

The data collected from the study supports the main thesis of this dissertation. Overall, participants found Machina easier, faster, more intuitive, and felt that it provided a more fluid interaction with the machine, facilitating experimentation and task implementation. Furthermore, most participants agreed that the capacity to interact in real-time with the device while learning about it helped them better understand how to control it. These findings constitute initial evidence that Enactive Robotics is a suitable model to help novice users in the creative fields, with little or no technical expertise, access the world of robot programming and control and create successful robotic systems.

The study also served to identify some challenges currently present on the technical implementation. Users manifested confusion around the instruction syntax currently available in Machina, and advocated for a system that would prevent them from requesting "erroneous" actions. Furthermore, the majority of the users expressed the desire for both systems to incorporate more feedback during program composition, especially in regard to the coordinate system of the robot and to potential pre-visualizations of forthcoming actions. These topics will be further discussed on the conclusions in the last chapter.

Finally, it is important to acknowledge the large number of independent variables present in this study. While the training sets where identical per system for all users, the tasks purposely had a small degree of openness: the location or orientation of the square where not specified, or how to place the object on the platform was left up to the user. This helped observe which choices users made, and how they differed between systems, in order to better understand how each tool biased a particular programming style. However, this also introduced a small level of inconsistency in the metrics, making some comparisons perhaps too general. Furthermore, while the study aims at testing the capacity of the conceptual model to provide a suitable cognitive tool towards robot programming, the model here tested is deeply entwined with the digital interface used to execute it. As a consequence, it could be argued that some of the features evaluated by the users might be attributable to the graphical user interfaces (GUI), rather than the interaction models they foster. Subsequent work should be conducted in studies that focus on specific aspects of the cognitive model here evaluated, as well as providing test frameworks for the principles of the enactive model, independent from particular GUI implementations.

7 Discussion and Future Work

7.1 Summary

This dissertation presented Enactive Robotics, a novel conceptual model for the design of concurrent machine control systems.

The motivation behind the contribution of this model is twofold. On the one hand, it addresses the steep learning curves that typical robot programming systems exhibit, making their use virtually inaccessible to the non-highly skilled and trained individual. On the other, it challenges the constrained interaction space that conventional offline machine control mechanisms provide, an obsolete model that has remained unchanged for the past half century.

The main goal of the model is to blur the traditional divide between programming machines and controlling them, fostering the development of systems where programs are *enacted* concurrently on the machine during the design phase. For that purpose, the model is designed around two fundamental principles:

- The capacity of external decision-making agents to concurrently interact with the system via universal, contextual and platform-agnostic requests named *actions*.
- The central role of a *state* model in the robotic system as the mediator between the high-level requests of the decision-making agents, and their low-level representations in machine terms.

To illustrate its capacity, several sample technical implementations of the Enactive Robotics model were presented in this work, focusing specifically on case of industrial robotic arms. These samples are part of the global *Robot Ex Machina* project —"Robot From the Machine"—, with the main members discussed in this dissertation being:

- *Machina.NET*, a self-contained, dependency-less C# library of code implementing the full architecture of the Enactive Robotics model. This project showed the capacity of the model to provide a powerful, robust, yet versatile framework for robot concurrent programming and control in one single environment, and is geared towards advanced users familiar with writing code-based applications.
- An ecosystem of applications designed to create highly modular robotic systems, whose character and communication schemes follow the principles of the Enactive Robotics model. The architecture of these systems is built around the centrality of the *Machina Bridge*, an UI-based application built with the .NET library at its core, and which provides a stand-alone *state* model to the robot accessible to external applications. For this matter, a collection of Bridge clients in different platforms and programming languages was also presented. Each client is designed to integrate in popular frameworks in the creative industry, and use the same *action-based* protocol to interface with the Bridge. These projects target entry-level, novel users with little to no prior robotic knowledge, and showcase the capacity of the model to be platform-agnostic and extensible to a variety of platforms under the same conceptual architecture.

The main thesis defended in this dissertation is that robot control systems built after the principles of the Enactive Robotics model exhibit two main advantages over traditional ones. The main benefit is that the model is designed to provide an easier, and more intuitive entry point to robot programming and control for novice users. The action-based interaction paradigm that it offers is based on the epistemological theories of Piaget, Bruner and Varela et al., providing an enactive rather than symbolic representation of the system, hence aiding the cognitive process that lead to understanding such a tangible medium.

In order to back this assertion, a controlled user study was conducted as part of this research work. A group of 20 volunteers with no prior experience in robotics was recruited from groups of creative students and professionals from the Boston metropolitan area. The main goal of the study was to compare the participants' performance and feedback after learning how to use an industrial robotic arm for the first time with an industry standard "easy, user-friendly" programming environment, as opposed to a system built after the principles of the Enactive Robotics model —the Machina Bridge in this case. Participants were asked to follow standard sets of training material for each system, and implement the same two tasks with the programming environments that they provided. Participants were surveyed at different stages of this process, and quantitative metrics were gathered throughout the experiment by an external observer. The data gathered in the study showed that participants consistently found Machina an easier and more intuitive tool to work with the robot, and they felt it provided a more fluid interaction with the machine.

This dissertation also argued that systems built following the Enactive Robotics model systematically provide a richer and deeper scope of interaction models and control paradigms than traditional offline programming, widening the range of possible robotic systems developable from it. To support this argument, a collection of projects built with the above-mentioned technical implementations was introduced in this work. The projects were hereby presented as case studies of the variety of author backgrounds, robotic applications and

interaction paradigms that the model can enable, and helped illustrate its versatility to adapt to different system architectures and application requirements. The projects were also used to frame a critical discussion around the capacity of the model to accommodate different levels of user skill, concurrency and agency in decision-making agents, further reinforcing the main thesis of this dissertation.

7.2 Challenges and Future Work

The work developed as part of this research also helped highlight potential challenges present in its current formulation, and identify future lines of work.

Enactive Robotics, as presented in this dissertation, is a high-level conceptual model for the design of concurrent machine control systems. In principle, the model is particularly suited to address machines whose execution run times significantly differ in scale from the digital regulatory systems that drive them; this is to say, any physical device that is "much slower than a computer", such as machines that perform motion in three-dimensional space or with moving parts. In the scope of this dissertation, the Robot Ex Machina project was contributed as an exemplar technical implementation of the model, focusing specifically on industrial robotic arms. The studies discussed in this dissertation around these particular machines support its main thesis, but additional research should be conducted to assess the *extensibility of the model* to accommodate other types of devices. There are countless families of mechanical actuators that could fall under this category: 3D printers, general CNC machines, drones, autonomous vehicles, Arduino-based systems, etc., all of them with different affordances and nuances. And while this work presents theoretical evidence of the capacity of the model to constitute a generalizable, platform-agnostic solution, further technical work would be beneficial in this direction, especially in how it could inform some of the conceptual principles behind the model.

The results of the controlled user study clearly identified areas in the technical implementation that required further work. Participants in the study unanimously advocated for a higher degree of feedback from the system. Novice users, and especially those without experience in numerical representation of three-dimensional space, manifested an elevated degree of confusion around the coordinate system of the robot, relying on iterations of trial and error in order to familiarize themselves with the environment and achieve their goals. And while trial and error is precisely a form of enactive learning, as predicated by the model, this cognitive approach is not necessarily in conflict with providing additional aids to foster *informed enaction*. Technical implementations of the model should incorporate enhanced forms of visual and textual feedback, such as literal inputs, robot diagrams, augmented reality overlays, etc., providing a platform to give users a higher degree of confidence over their actions.

In this sense, users also communicated significant uncertainty —sometimes even fear— about the concrete effect of their actions prior to executing them. The technical implementations discussed in this work provide an accessible entry point to use the robot as a machine "to-learn-with," but rely on direct interaction with that

particular machine in order to provide feedback about the effects of actions requested on it. This is basically to say that, the only way to learn about the effect of an action on a robot is to execute that precise action on it. This scenario can be daunting at first, especially for novice users using larger or faster robots. As discussed before, the Enactive model does not conflict with digitally-enhanced ways of providing further feedback about the effect of an action, such as simulation of the final state of the robot before execution, implementation of augmented reality techniques to previsualize motion, etc. The Enactive Robotics model is fundamentally grounded in the capacity of the state representation to provide feedback on the concurrent state of machine execution. However, the results of the user study are very clear in pointing out the necessity of increased feedback prior to execution, suggesting the possibility of incorporating additional *future state* feedback as a new principle at the core of the model. Further research should be conducted on how to extrapolate this notion to a high-level conceptual structure to fit with the general architecture of the model.

Finally, a case should be made about safety issues regarding real-time control of mechanical actuators. The tangible affordance of a robot to be used as an instrument to-learn-with also involves certain risks. Trial and error processes are problematic when the former leads to the latter. When debugging pure code, the worst-case scenario is a run-time error, or perhaps some data loss. But an error controlling a large, powerful robot may have quite expensive consequences on the machine itself, or even worse, become a hazard for the safety of the humans around it; these are clearly the origins of the uncertainties and fears manifested by the users. In principle, the state representation at the core of the Enactive model should have all the information necessary to implement *safety protocols* to perform analysis such as out-of-reach prevention, travel through singularities, collision checks, or protection for humans on the loop; such should prevent users from enacting erroneous or dangerous actions. The careful and comprehensive design of those protocols was outside the scope of this dissertation, and deserves a treatise of its own.

7.3 Reflections

When I started writing this dissertation —or more precisely, when I first began to think about its objectives— I was primarily concerned with the act of *digital making*. All those dialectics between designing and doing, materials and matter, hardware and software, programming and executing... Being a passionate advocate of technology, I was tremendously troubled about how it may had detached us from a romanticized ideal of traditional craft, represented by the glorious and seamless wholeness of human-tool-matter; I made it my quest to further use technology to reunite them again.

Very soon, my attention turned to the apparent source of the problem: digital fabrication and *machines that make*, such as 3D printers, CNC machines, and ultimately, industrial robotic arms as the closest equivalent to an ideal "all-purpose fabrication machine". Perhaps their inherent design was critically biasing our capacity to creatively interact with them. And while I believe some truth may lie behind this suspicion, I also believe that

machines, just like computers, have the potential to be "projective Rorschach machines" (Nelson 1975, p. 9): what you see in them, how they behave is just simply what comes out of their creator's mind.

Such intuition led me to examining in greater detail the regulatory systems that we use to control machines, or in other words, the design of *interfaces for making*. As programmable machines, the capacity of robots to make things cannot[35] extend beyond the capacity of humans to instruct robots how to make them. So, I dedicated years to researching, experimenting and, finally, proposing a new model for human-robot interaction, one that would hopefully promote creative exploration through digital tools, and democratize its practice. And hence, the result of this dissertation.

I profoundly believe in accessibility as one of the cornerstones of knowledge contribution in such an interconnected society as we live in. Lots of examples are discussed in this work about the potential use of industrial robots on construction sites, yet I do not necessarily believe they are the right type of machine for this context. But enabling exploration in these avenues may spark the interest of other thinkers, researchers and creative individuals to delve deeper into this field, and perhaps address the hardware challenges remaining to be solved. I humbly recognize my limitations to make significant contributions in this direction.

Similarly, it may appear to the reader of this document that it was my secret agenda to blame G-Code for all things I consider wrong in our relationship with fabrication machines; nothing further from reality. If G-Code has managed to stick around for more than half a century, there are multiple good reasons to justify it, and I commend their anonymous designers for such a feat. Instead, I am much more interested in exploring the alternative scenario: why would anyone want to control a 3D printer in real time? I think we still don't have a good answer to this question, but that is exactly my point. I firmly believe that if such was a standard, out-of-the-box possibility in the way 3D printers are interfaced with, people around the globe would immediately start finding novel and practical applications of this technique[36]. Such is my, perhaps naïve, faith in the creative nature of the human mind.

But no matter how much discussion around robots is to be found in this document, I would like to conclude by making a very bold statement: this dissertation is, and has always been, about *humans that make*. I believe that, as a civilization, we have gotten pretty good at making machines that make things. However, I believe that there is still a lot of work to be done on how to adequately connect them to our human brains, work that should be tackled jointly through biology, cognitive science, engineering and design.

In a world where the global conversation is dominated by the fear of robots and artificial intelligence rendering all of us useless, I still remain optimistic. I look forward to a future where robots are an integral part of our daily lives, and constitute a source of aid and reassurance, not dread. I look forward to a future where human-

[35] Yet.

[36] Although some of the case studies in this dissertation point to interesting reasons why humans-on-the-loop will open up interesting possibilities in human-mediated making with industrial robots.

robot interaction is so fluid and seamless, that it may make us acknowledge the very nature of our own humanity. I look forward to a future where the story behind the robotic students in Peek (2016) is not one of failure due to poor system integration, but a success story of human-robot collaboration enabled by robotic systems that systematically incorporate humans at the core of their decision-making mechanisms.

I hope that the modest contribution behind this work may serve as a catalyst towards that vision.

Appendix A: Case Studies Data

In this Appendix, full credits are presented for the projects described in the Case Studies chapter.

Drawing with Robots

Authors: Nono Martínez Alonso and Jose Luis García del Castillo.

Date: November 2017.

Location: Autodesk BUILD Space, Boston, MA, United States of America.

Mind Ex Machina

Tutors: Nono Martínez Alonso, Jose Luis García del Castillo and Panagiotis Michalatos.

Workshop Participants: Carla de Beer, Matt Jezyk, Sean Wallish, Adriana Sadun, Noni Pittenger, Jigar Solanki, Ivana Seizova, Randa Omar, Naveed Khan, Adriana Sadun, Matthew Spremulli and Chris Chekan.

Date: May 2017.

Location: SmartGeometry Conference, Toronto, Canada.

Project: Selfie Plot.

Author: Matt Jezyk.

Project: Exquisite Corpse

Author: Sean Wallish.

Project: Mediated Construction

Authors: Naveed Khan and Adriana Sadun.

Project: Slip Cast Turntable

Authors: Matthew Spremulli and Chris Chekan.

RobotSketch-RNN

Authors: Nono Martínez Alonso and Jose Luis García del Castillo.

Date: April 2018.

Location: Autodesk BUILD Space, Boston, MA, United States of America.

Real-Time Anatomy

Authors: Shang Liu, Zhiheng Jiao and Jichen Wang.

Instructors: J. Pertierra, A. Bidgoli, H. Wang.

Date: December 2018.

Location: Carnegie Mellon University, Pittsburgh, PA, United States of America.

Dereliction

Author: Jose Luis García del Castillo.

Date: June 2018.

Location: Digital Stone Project Workshop, Gramolazzo, Italy.

Material Systems: Digital Design and Fabrication

Instructors: Martin Bechtold and Jose Luis García del Castillo.

Date: Fall 2017.

Location: Harvard Graduate School of Design, Cambridge, MA, United States of America.

Project: Ceramic Arch

Authors: Lubin Liu, Yonghwan Kim and Mari Jo.

Project: Geodesic Dome

Authors: Maitao Guo, Ching-Che Huang, Anqi Huo and Jianing Zhang.

Project: Pappardelle Pillars

Authors: Andrew Bako, and Alexandru Vilcu.

Project: Soft Aggregate Jamming

Authors: Aurora Jensen, Iain Gordon and Peter Osborne.

Project: The Mixology Table

Authors: Jiho Sejung Song, Haeyoung Kim and HyeJi Yang.

Spatial + Responsive Print Trajectory

Authors: Sulaiman AlOthman, Hyeonji Im and Francisco Jung.

Advisors: Martin Bechthold and Jose Luis García del Castillo.

Date: Fall 2017 to Fall 2018.

Location: Harvard Graduate School of Design, Cambridge, MA, United States of America.

Interactive 3D Printing

Authors: Jose Luis García del Castillo, Sulaiman AlOthman and Hyeonji Im.

Date: March 2017.

Location: Harvard Graduate School of Design, Cambridge, MA, United States of America.

Robotic Hot Wire Cutting

Authors: Cesar Fragachan, Jianfei Chu, Melis Kucuktunc and Taizhong Chen.

Tutors: Alicia Nahmad Vasquez and Shajay Bhooshan.

Date: Summer 2018.

Location: Autodesk BUILD Space, Boston, MA, United States of America.

Teachable Machina

Authors: Nono Martínez Alonso and Jose Luis García del Castillo.

Date: April 2018.

Location: Autodesk BUILD Space, Boston, MA, United States of America.

Greenbuild Pavilion

Author: Hakim Hasan and Anish Reddy (Perkins + Will).

Date: Fall 2018.

Location: Autodesk BUILD Space, Boston, MA, United States of America, and Greenbuild International Conference and Expo, Chicago, IL, United States of America.

Automated Fabrication and Assembly in Construction

Authors: Ardavan Bidgoli, Sevki Topcu and Mehmet Bermek.

Date: Summer 2018.

Location: Autodesk BUILD Space, Boston, MA, United States of America.

Tight Squeeze Pavilion

Authors: Hakim Hasan, Jose Luis García del Castillo, Shajay Bhooshan, Nathan King and Gustav Fagerstrom.

Workshop Participants: Thomas Adams, Ahmed Elmaraghy, Elias Darham, Aled Lloyd, Ruggero Cipolla, Natalie Sham, Shayani Fernando, Francesca Falchieri, Carol Kan, Peng Song and Matthew Spremulli.

Date: August-September 2018.

Location: Autodesk BUILD Space, Boston, MA, United States of America, and NCCR/DFAB at the ETH Zürich, Switzerland.

Appendix B: Controlled User Study Data

Introductory Information

For consistency in context about the study, upon start of the test, each participant was provided the following set of information:

> Dear participant,
>
> Thank you very much for joining this study!
>
> You are about to participate in a study on Interfaces for Robot Programming. The general goal of this test is to better understand how to help people better control robots.
>
> During the course of this test, you are going to be asked to try two different programming interfaces, and use them to program the robot to do various tasks. At each step of this test, you will be informed of what to do and what to expect. The test will take approximately 3 hours to complete.
>
> During the course of the test, you are going to be observed, and some of those observations potentially logged. You will not be informed of what is being observed, to avoid a bias in the study or your actions. Please feel comfortable, act naturally, and at any moment just do whatever you feel you want to do; it is the best way to contribute to the study. Please keep in mind that the purpose of the study is to evaluate the tools, not the users. Also, your observations will be kept private, not shared with third parties, and only represented anonymously on potential reports of the study.
>
> You are strongly encouraged to "think aloud" during the test. This will greatly help the observer in his/her observations. You are also allowed to ask questions at any time during the test. The observer may choose to answer or not depending on how the answer may affect the study.
>
> It is very important that you know you are allowed to quit the test at any time, either partially or completely, without the need to justify yourself. If at any moment you feel uncomfortable or frustrated, you are welcome to quit what you are doing at that moment and move on to the next stage, or quit completely.
>
> Finally, upon completion of the test, you will be given full context of the details of the observation and the goals of the study.

Any questions?

Let's do it!

Cheat Sheets

Participants in the study were provided with "cheat sheets" for each system, highlighting the main instructions covered on each training set, and for participants to take personal notes.

For ***PolyScope***:

Program Robot > Installation > TCP Configuration

- X, Y, Z
- RX, RY, RZ
- Payload
- Center of Gravity

Program Robot > Program > Structure > Basic

- Move
  - > Command > MoveJ > Set Waypoint
  - > Command > MoveL > Set Waypoint
  - > Command > MoveP > Set Waypoint
  - > Command > MoveP > Add CircleMove

Program Robot > Program > Structure > Basic

- Wait
  - > Command > Time
- Set
  - > Command > Digital Output

NOTE: in this test, the gripper is set up on regular (non-tool) digital output number **0**, with ***true* for grip**, and ***false* for open**.

For ***Machina***:

Move(x, y, z);

MoveTo(x, y, z);

SpeedTo(value);

PrecisionTo(radius);

Rotate(vx, vy, vz, angle);

Wait(milliseconds);

WriteDigital(pinNum, true/false);

NOTE: in this test, the gripper is set up on regular (non-tool) digital output number **0**, with ***true* for grip**, and ***false* for open**.

Training Material

The UR Academy training used in the study can be found on https://academy.universal-robots.com/. A walkthrough of the videos was recorded for archival purposes, and can be found here:

https://www.youtube.com/playlist?list=PLvxxYImPCApV-5tlCFHBa8c91IiDv8nuh

The Machina training material used in the study can be found under:

https://www.youtube.com/playlist?list=PLvxxYImPCApXrS3qvgoowzsLJoVwnglQ-

Survey Data

A complete list of the responses to all surveys can be found in comma-separated values format files attached to the PDF digital version of this dissertation. Alternatively, the data can be found at:

http://www.garciadelcastillo.com/EnactiveRobotics_UserStudyData.zip

Bibliography

"A Third Industrial Revolution." *The Economist*, 21 Apr. 2012. www.economist.com/special-report/2012/04/21/a-third-industrial-revolution. Accessed 2 Feb. 2016.

ABB Robotics. "Software Downloads." new.abb.com/products/robotics/robotstudio/downloads. Accessed 23 Aug. 2018.

———. *Technical Reference Manual, RAPID Instructions, Function and Data Types*. 2013.

Aish, Robert. "Tools of Expression: Notation and Interaction for Design Computation." *Proceedings of the 29th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, The School of the Art Institute of Chicago, 2009, pp. 30-31.

Alberti, Leon B. *On the Art of Building in Ten Books*. Translated by Joseph Rykwert et al. MIT Press, 1988.

AlOthman, Sulaiman, Hyeonji C. Im and Jose L. García del Castillo. "Spatial Print Trajectory." *Robotic Fabrication in Architecture, Art and Design*. Springer, Cham, 2018.

Amtsberg, Felix, et al. "From Analysis to Production and Back Attempts and Results of Reusable Adaptive Freeform Production Strategies for Double Curved Concrete Construction Elements." *Robotic Fabrication in Architecture, Art and Design 2016*. Springer, 2016, pp. 304-315.

Anderson, Chris. *Makers: The New Industrial Revolution*. Random House Business, 2012.

Andreani, Stefano, et al. "Flowing Matter: Robotic Fabrication of Complex Ceramic Systems." *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, vol. 29, 2012.

Autodesk Inc. *DynamoBIM* [computer software]. dynamobim.org/. Accessed 17 Dec. 2018.

Bard, Joshua, Ardavan Bidgoli, and Wei Wei Chi. "Image Classification for Robotic Plastering with Convolutional Neural Network." *Robotic Fabrication in Architecture, Art and Design 2018*. Springer, 2018, pp. 3-15.

Baudisch, Patrick, and Stefanie Mueller. "Personal fabrication." *Foundations and Trends in Human–Computer Interaction*, vol. 10, no.3–4, 2017, pp. 165-293.

Bechthold, Martin. "The Return of the Future: A Second Go at Robotic Construction." *Architectural Design* vol. 80, no. 4, 2010, pp. 116-121.

Bechthold, Martin and Jose L. García del Castillo. "2018 Material Systems: Digital Fabrication and Design." *MaP+S Research Group*, Jan. 2018. research.gsd.harvard.edu/maps/portfolio/2018-material-systems-digital-design-and-fabrication/. Accessed 18 Dec. 2018.

Biggs, Geoffrey, and Bruce MacDonald. "A Survey of Robot Programming Systems." *Proceedings of the Australasian Conference on Robotics and Automation*. 2003.

Braam, David. *Cura* [computer software]. Ultimaker, 2014. github.com/Ultimaker/Cura.

Braumann, Johannes, and Sigrid Brell-Çokcan. "Parametric Robot Control, Integrated CAD/CAM for Architectural Design." *Proceedings of the 31th Annual Conference of the Association for Computer Aided Design in Architecture*, 2011, pp. 242-251.

Brell-Çokcan, Sigrid, and Johannes Braumann, eds. *Rob|Arch 2012*. Springer, 2013.

Bruner, Jerome S. *On Knowing: Essays for the Left Hand*. Harvard University Press, 1962.

———. *Toward a Theory of Instruction*. Harvard University Press, 1966.

Cangelosi, Angelo and Matthew Schlesinger. *Developmental Robotics: From Babies to Robots*. MIT Press, 2015.

Carpo, Mario. *The Alphabet and the Algorithm*. MIT Press, 2011.

Davis, Daniel. *Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture*. 2013. RMIT University, Ph.D. Dissertation.

Davis, Daniel, and Brady Peters. "Design Ecosystems: Customising the Architectural Design Environment with Software Plug-ins." *Architectural Design*, vol. 83, no. 2, 2013, pp. 124-131.

Davis, Stanley M. *Future Perfect*. 10th Anniversary ed. Addison-Wesley, 1996.

De Landa, Manuel. "Philosophies of Design: The Case of the Modeling Software." *Verb Architecture Boogazine: Authorship and Information* 1. Eds. Jaime Salazar et al. Actar Press, 2002, pp. 130-43.

Deleuze, Gilles, and Felix Guattari. *A Thousand Plateaus.* University of Minnesota Press, 1987.

Dijkstra, Edsger W. "Some Meditations on Advanced Programming." EWD-32, 1962. *E.W. Dijkstra Archive*. Center for American History, University of Texas at Austin, Austin, TX. Accessed 9 March 2015.

Dörfler, Kathrin, Florian Rist, and Romana Rust. "Interlacing: An Experimental Approach to Integrating Digital and Physical Design Methods." *Rob|Arch 2012.* Springer, 2013, pp. 82-91.

Elashry, Khaled, and Ruairi Glynn. "An Approach to Automated Construction Using Adaptive Programing." *Robotic Fabrication in Architecture, Art and Design 2014*. Springer, 2014, pp. 51-66.

Erickson, Thomas D. "Working with interface metaphors." *The Art of Human-Computer Interface Design*. Ed. Brenda Laurel. Addison-Wesley Pub., 1990, pp. 65-73.

Fernaeus, Ylva, Martin Jonsson, and Jakob Tholander. "Revisiting the Jacquard Loom: Threads of History and Current Patterns in HCI." *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012.

Friedman, Jared, Ahmed Hosny, and Amanda Lee. "Robotic Bead Rolling." *Robotic Fabrication in Architecture, Art and Design 2014*. Springer, 2014, pp. 83-98.

Froese, Tom, and Tom Ziemke. "Enactive Artificial Intelligence: Investigating the Systemic Organization of Life and Mind." *Artificial Intelligence,* vol. 173, nos. 3-4, 2009, pp. 466-500.

Gannon, Madeline. *Human-Centered Interfaces for Autonomous Fabrication Machines.* 2018. Carnegie Mellon University, Ph.D. Dissertation.

García del Castillo, Jose L. *Interfaces for Digital Making: Reflections at the Intersection of Creativity, Matter, Computers and Minds*. Self-published, 2016.

———. "Machina.NET: A Library for Programming and Real-Time Control of Industrial Robots." *Journal of Open Research Software,* 2019 (pending publication).

———. "Robot Ex Machina." *GitHub*, 2016. github.com/RobotExMachina. Accessed 17 Dec. 2018.

Gershenfeld, Neil. *Fab: The Coming Revolution on your Desktop - From Personal Computers to Personal Fabrication*. Basic Books, 2008.

Gibson, James J. "The Theory of Affordances." *Perceiving, Acting and Knowing: Toward an Ecological Psychology*. Eds. Robert E. Shaw and John D. Bransford. Lawrence Erlbaum Associates, 1977, pp. 67-82.

Gleick, James. *The Information: A History, a Theory, a Flood*. Vintage Books, 2012.

Gomoll, Kathleen. "Some Techniques for Observing Users." *The Art of Human-Computer Interface Design*. Ed. Brenda Laurel. Reading, MA: Addison-Wesley Pub., 1990, pp. 85-90.

Gramazio, Fabio and Kohler, Matthias. *The Robotic Touch: How Robots Change Architecture*. Park Books, 2014.

Gruber, Howard E. and J. Jacques Vonèche, eds. *The Essential Piaget: An Interpretive Reference and Guide*. Basic Books, 1977.

Ha, David, and Douglas Eck. "A Neural Representation of Sketch Drawings." arXiv preprint arXiv:1704.03477v4, 2017.

Hoare, C. A. R. *Hints on Programming Language Design.* Stanford Computer Science Report CS403, 1973.

Howe, Hartley E. "Teaching Power Tools to Run Themselves." *Popular Science,* Aug. 1955.

Im, Hyeonji C., Sulaiman AlOthman and Jose L. García del Castillo. "Responsive Spatial Print: Clay 3D Printing of Spatial Lattices Using Real-Time Model Recalibration." *Proceedings of the 38th Annual Conference of the Association for Computer-Aided Design in Architecture (ACADIA).* Universidad Iberoamericana, 2018.

Ingold, Tim. "The Textility of Making." *Cambridge Journal of Economics,* vol. 34, no. 1, 2009, pp. 91-102.

Ishii, Hiroshi, and Brygg Ullmer. "Tangible bits: towards seamless interfaces between people, bits and atoms." *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*. ACM, 1997.

ISO 6983-1: 2009. Automation Systems and Integration—Numerical Control of Machines—Program Format and Definitions of Address Words—Part 1: Data Format for Positioning, Line Motion and Contouring Control Systems. International Organization for Standardization, Geneva, Switzerland. www.iso.org/standard/34608.html

Isola, Phillip, et al. "Image-to-Image Translation with Conditional Adversarial Networks." arXiv preprint arXiv:1611.07004, 2016.

Jeffers, Michael. "Autonomous Robotic Assembly with Variable Material Properties." *Robotic Fabrication in Architecture, Art and Design 2016*. Springer, 2016, pp. 48-61.

Johns, Ryan L. *Mussel* [computer software], 2014. grasshopper3d.com/group/mussel. Accessed 17 Dec. 2018.

Jones, Byron, and Michael G. Kenward. *Design and analysis of cross-over trials*. Chapman and Hall/CRC, 2014.

Kay, Alan. "User Interface: A Personal View." *The Art of Human-Computer Interface Design*. Ed. Brenda Laurel. Addison-Wesley Pub., 1990, pp. 191-207.

Keating, Steven, et al. "A Compound Arm Approach to Digital Construction." *Robotic Fabrication in Architecture, Art and Design 2014*. Springer, 2014, pp. 99-110.

KUKA Roboter GmbH. *Software KR C2 I KR C3 Expert Programming*. 2003.

Landay, James A. "Technical Perspective: Design Tools for the Rest of Us." *Communications of the ACM,* vol. 52, no. 12, 2009, p. 80.

Laurel, Brenda. "What is an interface?" Introduction. *The Art of Human-Computer Interface Design*. Ed. Brenda Laurel. Addison-Wesley Pub., 1990, pp. xi-xvi.

Lialina, Olia. *Turing Complete User*. Oct. 2012. www.contemporary-home-computing.org/turing-complete-user/. Accessed 2 Feb. 2016.

Liberman, Zach, et al. "openFrameworks." 14 Nov. 2018, openframeworks.cc. Accessed 17 Dec. 2018.

Maeda, John. *Design by Numbers.* MIT Press, 2001.

Martínez Alonso, Nono. *Suggestive Drawing Among Human and Artificial Intelligences*. 2017. Harvard University, M.Des. Thesis.

Matarić, Maja J. *The Robotics Primer*. MIT Press, 2007.

McCarthy, Lauren. *p5.js* [computer software]. p5js.org. Accessed 17 Dec. 2018.

McCullough, Malcolm. *Abstracting Craft: The Practiced Digital Hand*. MIT Press, 1996.

McGee, Wes, and Monica Ponce de Leon, eds. *Robotic Fabrication in Architecture, Art and Design 2014.* Springer, 2014.

McNeel, Robert. *Rhinoceros: Modeling Tools for Designers* [computer software]. Robert McNeel & Associates, 1993. www.rhino3d.com.

Mellis, David, et al. "Arduino: An Open Electronics Prototyping Platform." *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2007.

Merleau-Ponty, Maurice. *Phenomenology of Perception*. Routledge & Kegan Paul, 1962.

Mitchell, William J., and Malcolm McCullough. *Digital Design Media*. 2nd ed. Van Nostrand Reinhold, 1995.

Moore, Dan, and Madeline Gannon. "ofxRobotArm." *GitHub*, 20 July 2016, github.com/CreativeInquiry/ofxRobotArm. Accessed 17 Dec. 2018.

Mountford, Joy S. "Tools and Techniques for Creative Design" *The Art of Human-Computer Interface Design*. Ed. Brenda Laurel. Addison-Wesley Pub., 1990, pp. 17-30.

Mueller, Stefanie. *Interacting with Personal Fabrication Devices*. 2016. University of Potsdam, Ph.D. Dissertation.

Mueller, Stefanie, Pedro Lopes, and Patrick Baudisch. "Interactive Construction: Interactive Fabrication of Functional Mechanical Devices." *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2012, pp. 599-606.

Mumford, Lewis. *Technics & Civilization*. University of Chicago Press, 2010.

"Myro, a Fun-Loving Robot." *Nokimono*, www.nokinomo.com/en/myro/index.html. Accessed 17 Dec. 2018.

Nelson, Ted. *Computer Lib / Dream Machines*. Expanded ed. Self-published, 1975.

———. "The Right Way to Think About Software." *The Art of Human-Computer Interface Design*. Ed. Brenda Laurel. Addison-Wesley Pub., 1990, pp. 235-243.

Nicol, Anne. "Interfaces for Learning: What Do Good Teachers Know that We Don't?" *The Art of Human-Computer Interface Design*. Ed. Brenda Laurel. Addison-Wesley Pub., 1990, pp. 113-122.

Noble, David F. *Forces of Production: A Social History of Industrial Automation*. Transaction Publishers, 2011.

Norman, Donald. *The Design of Everyday Things*. Rev. an exp. ed. Basic Books, 2013.

Pan, Zengxi, et al. "Recent Progress on Programming Methods for Industrial Robots." *Robotics and Computer Integrated Manufacturing*, vol. 28, no. 2, 2012, pp. 87-94.

Papert, Seymour. *Mindstorms*. 1980. 2nd ed., Basic Books. 1993.

Papert, Seymour, and Idit Harel. "Situating Constructivism." Introduction. *Constructionism: Research Reports and Essays, 1985-1990*. Eds. Seymour Papert and Idit Harel. Ablex Pub., 1991.

Payne, Andrew O., and Jason K. Johnson. "Firefly: Interactive Prototypes for Architectural Design." *Architectural Design*, vol. 83, no. 2, 2013, pp. 144-147.

Peek, Nadya. *Making Machines that Make: Object-Oriented Hardware Meets Object-Oriented Software*. 2016. Massachusetts Institute of Technology, Ph.D. Dissertation.

Peng, Huaishu, et al. "RoMA: Interactive Fabrication with Augmented Reality and a Robotic 3D Printer." *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018.

Petzold, Charles. *Code: The Hidden Language of Computer Hardware and Software*. Microsoft Press, 2000.

Pfeifer, Rolf and Christian Scheier. *Understanding Intelligence*. MIT Press, 1999.

Processing Foundation. "Jose Luis Garcia del Castillo - Processing Community Day 2017." *YouTube*, talk by Jose L. García del Castillo, 16 Jan. 2018, youtu.be/RK254vFB3KM.

Quigley, Morgan, et al. "ROS: An Open-source Robot Operating System." *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, 2009.

Raspall, Felix. *Design with Material Uncertainty: Responsive Design and Fabrication in Architecture.* 2015. Harvard University, D.Des Dissertation.

Raspall, Felix, Felix Amtsberg, and Stefan Peters. "Material Feedback in Robotic Production." *Robotic Fabrication in Architecture, Art and Design 2014*. Springer, 2014, pp. 333-345.

Raymond, Eric. "The Cathedral and The Bazaar." *Knowledge, Technology & Policy*, vol. 12, no. 3, 1999, pp. 23-49.

Reas, Casey and Ben Fry. *Processing: A Programming Handbook for Visual Designers and Artists.* MIT Press, 2007.

Reinhardt, Dagmar, et al. "TriVoc." *Robotic Fabrication in Architecture, Art and Design 2014*. Springer, 2014, pp. 163-180.

Reinhardt, Dagmar, et al., eds. *Robotic Fabrication in Architecture, Art and Design 2016.* Springer, 2016.

Rheingold, Howard. "An interview with Don Norman" *The Art of Human-Computer Interface Design*. Ed. Brenda Laurel. Addison-Wesley Pub., 1990, pp. 5-10.

Richards, Neil M. and William D. Smart. *How Should the Law Think About Robots?* 10 May 2013. ssrn.com/abstract=2263363. Accessed 15 Dec. 2018.

RoboDK Inc. *RoboDK* [computer software]. www.robodk.com. Accessed 17 Dec. 2018.

Rutten, David. *Grasshopper: Algorithmic Modeling for Rhino* [computer software]. Robert McNeel & Associates, 2009. www.grasshopper3d.com.

Schön, Donald A. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, 1984.

Schumacher, Patrik. "Parametricism: A New Global Style for Architecture and Urban Design." *Architectural Design*, vol. 79, no. 4, 2009, pp. 14-23.

Schwartz, Thibault. "HAL: Extension of a Visual Programming Language to Support Teaching and Research on Robotics Applied to Construction." *Rob|Arch 2012*. Springer, 2013, pp. 92-101.

Seibold, Zach, et al. "Ceramic Morphologies: Precision and Control in Paste-Based Additive Manufacturing." *Proceedings of the 38th Annual Conference of the Association for Computer Aided Design in Architecture*, 2018, pp. 350-357.

Sennett, Richard. *The Craftsman*. Yale University Press, 2008.

Shahmiri, Fereshteh, and Jeremy Ficca. "A Model for Real-Time Control of Industrial Robots." *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, vol. 33. Vilnius Gediminas Technical University, Department of Construction Economics & Property, 2016.

Sharif, Shani, T. Russell Gentry, and Larry M. Sweet. "Human-Robot Collaboration for Creative and Integrated Design and Fabrication Processes." *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, vol. 33. Vilnius Gediminas Technical University, Department of Construction Economics & Property, 2016.

Shneiderman, Ben. "Direct Manipulation: A Step Beyond Programming Languages." *Computer,* vol. 16, no. 8, 1983, pp. 57-69.

Smith, Cyril S. "Matter versus Materials: A Historical View." *Science* 162, 1968. Rpt. in *A Search for Structure: Selected Essays on Science, Art, and History*. Ed. Cyril Stanly Smith. MIT Press, 1981, pp. 112-126.

Smith, David C. *Pygmalion: A Creative Programming Environment*. Stanford University AI Memo 260, CS Dept. Stan-CS-75-499, Jun. 1975.

Smith, David C., et al. "Programming by Example: Novice Programming Comes of Age." *Communications of the ACM*, vol. 43, no. 3, 2000, pp. 75-81.

Snooks, Roland, and Gwyllim Jahn. "Closeness: On the Relationship of Multi-Agent Algorithms and Robotic Fabrication." *Robotic Fabrication in Architecture, Art and Design 2016*. Springer, 2016, pp. 218-229.

"Stack Overflow Developer Survey 2018". Stack Overflow, 2018, insights.stackoverflow.com/survey/2018/. Accessed 17 Dec. 2018.

"Standard ECMA-404: The JSON Data Interchange Syntax." *ECMA International,* 2nd. Ed, Dec. 2017.

Steiner, Hans-Christoph. "Firmata: Towards Making Microcontrollers Act Like Extensions of the Computer." *NIME*. 2009, pp. 125-130.

Stewart, John, Olivier Gapenne and Ezequiel A. Di Paolo. *Enaction: Toward a New Paradigm for Cognitive Science.* MIT Press, 2010.

"Teachable Machine." *GitHub*, 28 April 2017, github.com/googlecreativelab/teachable-machine. Accessed 17 Dec. 2018.

Thompson, Evan. *Mind in Life: Biology, Phenomenology and the Sciences of Mind*. Harvard University Press, 2007.

Toffler, Alvin. *The Third Wave*. Bantam Books, 1981.

Universal Robots. "PolyScope Manual, Version 3.7.0." 2018. s3-eu-west-1.amazonaws.com/ur-supportsite/44371/Software_Manual_en_Global.pdf. Accessed 02 Nov. 2018.

———. "Robotics as it Should Be: Simple, Flexible, Affordable." 2015. www.universal-robots.com/media/8683/ur_brochure_us.pdf. Accessed 02 Nov. 2018.

———. "The URScript Programming Language, Version 3.7.0." 2018. s3-eu-west-1.amazonaws.com/ur-support-site/44407/scriptManual_en.pdf. Accessed 02 Nov 2018.

———. UR Academy. www.universal-robots.com/academy/ Accessed 02 Nov. 2018.

Varela, Francisco J., Evan Thompson and Eleanor Rosch. *The Embodied Mind: Cognitive Science and Human Experience*. 1991. Rev. ed., MIT Press, 2016.

Vasey, Lauren, Iain Maxwell, and Dave Pigram. "Adaptive Part Variation." *Robotic Fabrication in Architecture, Art and Design 2014*. Springer, 2014, pp. 291-304.

Visual Components Oy. *Visual Components* [computer software]. www.visualcomponents.com. Accessed 17 Dec. 2018.

Vogler, Verena. "Taco – ABB Robot Programming for Grasshopper." *Rhino News*, McNeel, 3 Mar. 2016, blog.rhino3d.com/2016/03/taco-abb-robot-programming-for.html. Accessed 17 Dec. 2018.

Von Neumann, John. "First Draft of a Report on the EDVAC." *Annals of the History of Computing, IEEE*, vol. 15, no, 4, 1993, pp. 27-75. First pub. in 1945.

Wake, Warren Kent. *TIGRIS: A Tool-Structured Interface and Graphic Interaction System for Computer-Aided Design*. 1992. Harvard University, D.Des. Dissertation.

Watson, Theo, et al. "Mimic." *Design I/O*, design-io.com/projects/Mimic. Accessed 17 Dec. 2018.

"WebSocket." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 22 July 2004. Web. 1 Dec. 2018, en.wikipedia.org/wiki/WebSocket. Accessed 17 Dec. 2018.

Webster, Barron. "Designing (and Learning From) a Teachable Machine." *Google Design*, 17 April 2018, design.google/library/designing-and-learning-teachable-machine/. Accessed 17 Dec. 2018.

Willis, Karl D. et al. "Interactive Fabrication: New Interfaces for Digital Fabrication." *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*. ACM, 2011, pp. 69-72.

Willmann, Jan, et al., eds. *Robotic Fabrication in Architecture, Art and Design 2018.* Springer, 2018.

Wijnen, Bas, et al. "Free and Open-Source Control Software for 3-D Motion and Processing." *Journal of Open Research Software*, vol. 4, no. 1, 2016.

Wirth, Niklaus. "On the Design of Programming Languages." *IFIP Congress,* vol. 74, 1974.

Xu, Xun W, and Stephen T. Newman. "Making CNC Machine Tools More Open, Interoperable and Intelligent—A Review of the Technologies." *Computers in Industry*, vol. 57, no. 2, 2006, pp. 141-152.

Yuan, Philip F., et al. "Robotic Multi-dimensional Printing Based on Structural Performance." *Robotic Fabrication in Architecture, Art and Design 2016.* Springer, 2016, pp. 92-105.