A SOFTWARE ENVIRONMENT FOR DESIGNING THROUGH ROBOTIC FABRICATION

Developing a graphical programming toolkit for the digital design and scaled robotic fabrication of high rises

Jason LIM, Fabio GRAMAZIO and Matthias KOHLER
ETH Zurich, Zurich, Switzerland
jason.lim@arch.ethz.ch, gramazio@arch.ethz.ch, kohler@arch.ethz.ch

Abstract. The term “robot” was born from a play written almost a century ago. Today robotic fabrication has become an emerging topic in architectural research. As architects work with these technologies, they are challenged with writing a different kind of play: here robots are the actors and the physical materialization of a design is their performance. However current Computer Aided Design (CAD) packages do not provide native robot programming functionalities which architects require to plan and orchestrate these fabrication process. To address this limitation, a Python library for robot programming is written. It is referenced by a toolkit of custom components developed to extend a graphical programming environment commonly used for architectural design. The empirical development of these software tools takes place in the context of a design studio investigating the subject of the high rise. The tools are tested in a workflow that involves the digital design and scaled robotic fabrication of high-rise housing. This paper discusses the considerations underlying the toolkit’s design, the outcomes of its use in the studio, and its impact on the creative design process.

Keywords. Robotic fabrication; Architectural model; Software tools; High Rise design; Creative computational design.

1. Introduction

In 1920, during a period of widespread optimism in technology, Karel Čapek wrote a play from which the term “robot” was born. The robots in his story were artificial workers created to replace human labour in the serial production of goods. Almost a century later, robotic fabrication has emerged as a topical area of
research in the field of architecture. Here, industrial robots are used for the production of digitally described building components that are often highly individualized.

This process requires the authorship of both design data, usually encapsulated in three-dimensional digital models, and production data, in the form of robot programs describing fabrication steps. However Computer Aided Design (CAD) packages used by architects nowadays only facilitates the digital design process. To plan and control the subsequent fabrication process, separate offline programming software provided by robot manufacturers is often used instead.\(^2\) Thus at the software level, design and production are defined as being distinct tasks.

In order to conceive of design and production as a comprehensive process\(^3\), the lack of robot programming and control functionalities in current CAD packages needs to be addressed. This paper describes the development of a custom Python library called \textit{Your} which provides these functionalities for the Rhinoceros/Grasshopper\(^4\) CAD platform. Using this extended environment, architects are now able to explore designs and plan production processes in a seamless manner. This paper will discuss the use of these new tools in the context of a design studio, as well as their impact on the creative design process.

2. Context of the Robotic Fabricated High Rise

This past decade has witnessed increasing experimentation with robotic fabrication within the architectural field. Thus far such work has concentrated on elements of a building and been limited in size. To meet the challenge of applying robotic fabrication at the large scale, the Chair of Architecture and Digital Fabrication\(^5\) at ETH Zurich set up a research group at the Future Cities Laboratory\(^6\). The primary topic of investigation is the design of robotic fabricated high rises.

An initial strategy of scaled robotic fabrication is pursued in order to address the high rise building as a comprehensive design subject. Here, physical models are directly fabricated from digital data, and play an important role in the conception and testing of architectural ideas. This approach offers considerable benefits for high rise design. Physical models provide direct and intuitive feedback about structural behaviour. This is significant for high rises as the resolution of both vertical and horizontal loads\(^7\) is an important design task. The principle of additive assembly, which the model building process is based on, applies to 1:1 high rise construction as well.\(^8\) Constructive concepts can thus be developed in the early design process and tested through robotic building.

An architecture studio exploring the subject of contemporary high rises in Singapore forms a key part of the research project. A workflow is adopted that
involves the digital design and scaled robotic fabrication of high rise housing. The model building process is based on the additive assembly of parts through pick and place operations. To facilitate this prototypical design methodology, a set of bespoke software tools called Your was developed for use in the studio.

3. Software Development

Your consists of a package of Python modules that adds the following functionalities to the Rhinoceros/Grasshopper CAD platform: a wrapper for the scripting language of the robots used, utility transformation functions, kinematics solvers and computer-robot communication. This package can be referenced through the script editor in Rhinoceros or Python components in Grasshopper; either environment can now be used for authoring robot programs.

For the design studio, Grasshopper was selected as the underlying environment for design and fabrication. This was due to the following considerations: firstly many students did not have prior programming experience and were unfamiliar with algorithmic approaches to design, and secondly the physical fabrication setup was expected to be frequently modified by students over the course of the studio. Here the visual programming and parametric nature of Grasshopper are advantages. Algorithmic logics can be implemented in a visual way by connecting components; this is accessible to non-programmers as it does not require writing code. By assigning fabrication parameters that are adjustable, Grasshopper definitions can be adapted to changing physical setups without the need for restructuring.

Based on this decision, a toolkit of Python scripting components, each referencing Your, was further developed for use in Grasshopper (Figure 1). The aims of the toolkit are twofold. First, it has to be easy to use so that robotic building is...
immediately accessible during the early design phase. Second, it has to be adaptable in order to allow experimentation with custom developed robotic processes.

The toolkit components are grouped into: setup, commands (interface, motions and custom actions), robot communications and kinematics. Setup components allow physical information such as the model building base to be stored. Command components wrap default URScript functions; inputs are given in intuitive to understand formats, for example orientation data as planes rather than axis-angles, and the output is auto-generated UR scripts. Robot communication components handle the sending and receiving of data through sockets, while kinematics components provide a way to visualize and simulate the robot. These components provide the necessary functionalities for a complete robot program to be created and sent. With tasks such as code generation, socket communications and coordinate system transformations handled behind the scenes, designers can focus on the logical assembly of components to define a program.

Other projects such as KUKA|prc and HAL also provide robot programming functionalities in Grasshopper through custom components. A distinguishing feature of the approach taken here is the emphasis on end-user modification. This was a consideration from the outset as having closed and overly specific components would have inhibited experimentation with bespoke robotic processes. All components provided in the toolkit are therefore open with internal code exposed. They serve as example blueprints, which users refer to when developing their own eventual components.

4. Studio Outcomes

During the design process, students create grasshopper definitions that contain both design and production data. These definitions are structured in four general parts (Figure 2). First, design information is generated and then parsed to get raw input data for command components; this involves the use of regular Grasshopper components. Second, information regarding the physical fabrication setup is added; setup components from the toolkit are used in this step. Third, detailed robot instructions are generated describing action and motion sequences; robot command components from the toolkit are predominantly used here. Finally, instructions are sent to the robot for execution with the help of communication components.

Two projects, which illustrate the main approaches emerging from the studio, are selected for further discussion (Figure 3). They both employ a constructive system involving the assembly of horizontal and vertical parts. However in the first case, each part is unique with shape information added by the laser cutter. In the second case, standard parts are used and subsequent information is added through a robotic deformation process.
A SOFTWARE ENV. FOR DESIGING THROUGH ROBOTIC FABRICATION

Figure 2.

Figure 3.
4.1. DESIGNING WITH MANY UNIQUE PARTS

A main intention of Team Bahru’s project was to explore the design potentials of a production process that enables every part of a building to be unique. Several tower versions were constructed that were assembled out of laser-cut cardboard walls and slabs.

Design and robot programming were separated in two Grasshopper definitions. In the first, an associative model was developed to explore the design of the high rise. Selected versions were baked in Rhinoceros as three-dimensional geometry. A subsequent Grasshopper definition for robot programming was created that referenced this baked geometry. From this, two-dimensional drawings of walls and floors were generated by a custom developed python scripting component, and organized into cut-sheets for the laser-cutter. Cut-sheets were placed in a physical feeder and provided picking information for the robot. Baked geometry provided raw information for placing operations.

For earlier tower versions, pick and place components from the toolkit were mainly used. However with the shift towards unique parts in later designs, robot motion trajectories had to be planned in greater detail to avoid issues of collision and imprecision. Pick and place components, which encapsulated pre-defined safety motions and actuator actions, were modified by the students. Components that provided lower level functions, such as linear motion, were assembled together to describe more complex building tasks. The eventual program was split into chunks and sent; parts were picked, placed and glued by the robot storey-by-storey. During the fabrication process, parameters such as the height of the model building base or wait times for gluing could be adjusted to account for material tolerances and building inaccuracies.

Two key design ideas of a branching system of twisted shear walls and a continuous central void were studied through a series of models. Tower fragments were built to test the positioning of walls and amount of twist that could be implemented without losing structural integrity. Different void shapes were studied to evaluate the effects on day-lighting and interior views. Based on the feedback gained from these physical models, the digital models were subsequently adjusted and refined, before being materialized again.

4.2. DESIGNING WITH ONE FOLDED PART

The design approach undertaken by Team Rochor was to derive maximum complexity from a simple material folding process. This process became a key design driver and determined the formal vocabulary of the high rise. Two 1:50 models were fabricated using a floor and folded-wall constructive system.
The team experimented with several modeling materials including cardboard and aluminum. Rules for folding were established, and the range of forms produced from their different permutations catalogued. Folded walls were produced in this manner; they were positioned based on rules that ensure the direct vertical transfer of loads to the immediate walls below. Design solutions were arrived at that negotiated between the formal possibilities offered by the folding process and the positional constraints imposed by the structural logic. A parametric model was developed to describe this.

The model assembly process was composed of repeated pick, fold and place operations. Cardboard or aluminum rectangular pieces were stacked on a simple feeder; as they were all standardized, one picking action could be programmed and repeated. The parametrically defined high rise design provided the raw information for placing. The main complexity of the robot program came from the folding operation. Students determined the precise robotic motions needed for folding a desired shape and encapsulated the logic in a cluster of basic toolkit components. The robot program was developed in relation to a customized physical setup, which included an actuated clamping station that was used in the folding process. To account for different material stiffness and malleability, additional parameters were added that allowed the speed and amount of over-folding to be adjusted.

In this project, the inherent stability of the folded wall meant that the model could be stacked without glue. The robot program was sent in chunks and after every level was built, the fabrication process was paused. The physical model could be evaluated at this stage and dissatisfactory parts disassembled by hand. Design parameters were tweaked, and this was automatically propagated to the robot program. The script for the previous disassembled level is then replayed. In this manner, the team was able to change the high-rise design during the building cycle itself.

5. Evaluation of Toolkit

The toolkit was empirically developed and tested over the course of the studio. Based on student feedback and direct observation of its use, the toolkit was iteratively refined to address identified bugs and missing functionalities. While the original toolkit given out contained six components, this number expanded to twenty by the end of the studio.

The first aim of the toolkit was to be easy to use so that robot programming would be accessible. In a three day workshop conducted at the start of the studio, students were introduced to toolkit for the first time and were able to successfully program the robotic fabrication of models up to a meter tall by its conclusion. In
interviews conducted at the end of the studio, students gave positive feedback when asked to evaluate the effectiveness of the toolkit in easing robot programming. The following responses were offered: the plug and play approach was considered to be intuitive to understand and accessible as it required no prior programming knowledge; the visual representation of robot programs as connected graphs rendered their structure and sequence more legible and were thus easier to edit; the auto-generation of robot scripts through components was a positive feature as it guaranteed code that was error free and executable by the robot.

A second objective was to enable architects to develop bespoke fabrication processes to design with. In this regard, the openness and extensibility of the toolkit enabled students to explore a range of robotic processes without being unduly constrained. This took place in three ways and usually in accordance with the development of a physical setup. The first approach involved assembling basic motion and action components to define more complex sequences. Second, students adapted pre-existing toolkit components by selectively editing and adding lines of code within it. Finally, custom scripts that reference Your library were written from scratch within Python components. In one example, a component was developed for a process involving plastic strip bending with a hot air gun. As projects evolved to incorporate custom fabrication processes, students began to actively modify the toolkit and the development of such components became commonplace.

6. Conclusion

The versatility of general purpose industrial robots has enabled them to be adapted for use in architectural design. This versatility is derived in large part from their programmable nature. This paper makes a case for robot programming interfaces to be directly embedded in the Computer Aided Design environments used by architects. Through the example of Your toolkit, it is suggested that graphical programming based interfaces offer benefits in terms of ease of use and adaptability.

By extending CAD environments in this direction, digital design and physical production can be merged from the outset. The planning of robotic production processes takes place concurrently with design exploration and the two become mutually informing. The production process enables the creation of physical models that are reproducible, easily varied and serial. Architectural ideas explored in the digital medium can now be directly evaluated in the added physical dimension through iterative model-building. At the same time physical production processes impose formal and organizational constraints that help define the space of design possibilities. In this mode of working, an architect is no longer just a form-giver, but takes on an expanded role as designer and orchestrator of production processes for realizing form.
A SOFTWARE ENV. FOR DESIGING THROUGH ROBOTIC FABRICATION

Endnotes

1. The play “Rossums Universal Robots” was a cautionary story about a servile force of robot workers that eventually revolted against their human creators.
2. In offline programming a user directly creates a robot program and determines its structure, flow and content – this is usually done through a computer. An example of such software is RobotStudio by ABB.
3. (Willman et al., 2012, p. 13).
7. A characteristic of a high rise is that it faces significant horizontal loads, in addition to vertical ones (Grohmann and Kloft, 2012, p. 77).
8. In the case of Singapore, the predominant method of high rise construction is the assembly of pre-fabricated reinforced concrete panels.
9. URScript is the proprietary scripting language for the Universal Robots used.
10. Orientation information is generally represented in one of the following formats: matrix, axis-angle notation and quaternions.
11. KUKA Parametric Robot Control (PRC) is a plugin for Grasshopper for programming KUKA robots (Brell-Cokcan and Braumann, 2010).
12. HAL is a plugin for Grasshopper for programming ABB robots (Schwartz, 2012).
13. This was a student project by Florence Thonney, Pascal Genhart, Patrick Goldener, and Tobias Wullschleger.
14. Baking is the operation of adding geometrical objects, created in Grasshopper, to a Rhinoceros document. The object’s associative relationships are severed.
15. This was a student project by Martin Tessarz, Sebastian Ernst, Silvan Strohbach and Sven Rickhoff.
16. This was a student project by Sylvius Kramer and Michael Stünzi.

Acknowledgements

We would like to thank Dr. Jan Willmann and Dr. Silke Langenberg for their invaluable feedback on this paper. In addition, this work is embedded within a larger research project at FCL and would be impossible without the close support of the following team members: module coordinator Michael Budig and fellow researchers Willi Lauer, Norman Hack and Raffael Petrovic. Finally, we would like to express our sincere gratitude to the following students: Alvaro Romero, Fabienne Waldburger, Florence Thonney, Martin Tessarz, Michael Stünzi, Pascal Genhart, Patrick Goldener, Sebastian Ernst, Silvan Strohbach, Sven Rickhoff, Sylvius Kramer and Tobias Wullschleger.

References


