Parametric Robot Control

INTEGRATED CAD/CAM FOR ARCHITECTURAL DESIGN

Johannes Braumann
Vienna University of Technology

Sigrid Brell-Cokcan
Vienna University of Technology

ABSTRACT

Robots are gaining popularity in architecture. Snøhetta has recently purchased their own industrial robot, becoming one of the first architectural offices to adopt robot technology. As more and more architects are exposed to robotic fabrication, the need for easy interoperability, integration into architectural design tools and general accessibility will increase. Architects are discovering that industrial robots are much more than kinematic machines for stacking bricks, welding or milling - they are highly multifunctional and can be used for a huge variety of tasks. However, industry standard software does not provide easy solutions for allowing direct robot control right from CAAD (Computer Aided Architectural Design) systems. In this paper we will discuss existing methods of programming industrial robots, published architectural results (Gramazio and Kohler 2008) and the design of a new user interface that allows intuitive control of parametric designs and customized robotic mass production, by integrating CAM (Computer Aided Manufacturing) functions into CAAD.

Keywords: robot programming, parametric design, mass customization, grasshopper component design, fabrication, robot milling, digital architecture

1 Introduction

The rapid development of CAAD software in the last few decades allows architectural forms to be parametrically controlled **(Figure 1)**, resulting in automatic generation of design variants or overall design management (Eastman et al. 2008). Realizing parametric designs requires flexible processes that can fabricate high volumes at reasonably low costs, referred to as *mass customization* (Piller 2004; Ramani et al. 2004).

Architects want to control the construction and fabrication process of their own designs, the *machinic processing* (Gramazio & Kohler, Menges, Scheurer in Kolarevic 2008; Iwamoto 2009). One problem arises as soon as (parametric) freeform designs are to be materialized: architects do not have appropriate software to control the very end of the overall design process: *fabrication*. Leading CAAD developers have so far not provided adequate solutions for directly linking different types of machines to CAD systems. The building industry involved in freeform architecture has overcome this deficit by developing their own "in-house" custom-made software tools. Therefore high-end geometry and fabrication consulting in architecture is rapidly becoming a new specialized core business run by computer scientists and mathematicians (refer to companies such as Evolute (Eigensatz et al. 2010) or design-to-production (Scheurer 2010)). In our previous research we have shown the advantages of production-immanent design and realtime feedback to evaluate not only the design of a parametric object, but also its manufacturing related properties (Brell-Cokcan and Braumann 2010).

In this paper we propose the use of a graphical algorithm editor as a new interface for an industrial robot. The result is user-customized CAAD software where machine constraints are included as additional design parameters directly generating robot control data files, to be used for mass customization.

2 Existing Methods for Programming a KUKA Robot

In general, there are two distinctive strategies for programming industrial robots: Online/automatic **(Figure 2)** and offline/manual programming **(Figure 3)** (Biggs and MacDonald 2003). Online programming is used mostly for industrial applications such as loading and welding, where points are "taught" individually via the KCP (KUKA Control Panel) or in the virtual robot environment KUKA SimPro. In the field of architecture, where the digital CAD/CAM workflow is important, online programming is rarely used.

2.1 ROBOT OFFLINE PROGRAMMING

Robot offline programming is performed at an external computer outside the robotic cell and is similar to the traditional way of working with CNC (Computer Numeric Control) machines, (i.e., moving data from CAD to CAM software) and then as NC (Numeric Control) code, also known as G-Code, to a CNC milling machine. Additionally, robot milling requires the use of a kinematic post-processor/simulator, in our case KUKA SimPro/CAMRob. This results in a digital workflow involving at least three software tools by different developers. "Realtime" design and production feedback that allows an evaluation of the fabrication properties throughout the workflow is missing.

Previous research shows the possibility of streamlining the digital CAD-CAM-Robot workflow by using parametric design tools for robot milling (Brell-Cokcan et al. 2009). To further optimize offline programming for a KUKA robot we will demonstrate in Section 4 the design of the custom KUKA|prc (parametric robot control) component within Grasshopper, a parametric design plug-in for the CAD software Rhinoceros.

2.2 KUKA ROBOT LANGUAGE CODE

The KUKA Robot Language (KRL) is a proprietary programming language with similarities to Pascal and is used to control KUKA robots. Unlike G-Code, it does not contain just tool and machine movement commands, but can also declare variables and work with conditional clauses. While the KRL documentation (KUKA Robotics 2003) explains a set of commands, it does not provide detailed information on the required structure of a proper control data file in

Fig. 1

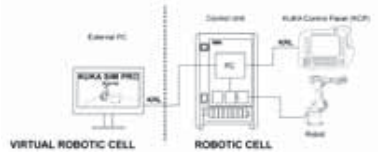

Fig. 2

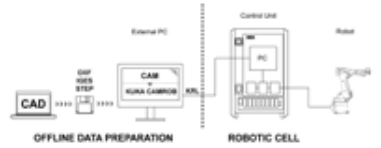

Fig. 3

Figure 1. *Spencer Dock Bridge: Parametric freeform design, CNC-milled formwork - Image courtesy of Nedcam*

Figure 2. *Online programming via KUKA Sim Pro (left) or via the KUKA Control Panel (right)*

Figure 3. *Existing offline data workflow from CAD to an industrial robot*

Fig. 4

KRL (Mühe et al. 2010). We therefore had to reverse-engineer the structure of KRL files created by the kinematics postprocessor KUKA SimPro/CAMRob. Basically, a Numeric Control file for KUKA consists of two separate files: A **.dat* file where variables such as the home position are stored and a **.src* file which contains the various commands, like movement commands, loops, sensor queries et cetera. There are various ways of giving movement commands to a robot, the most common ones are the following (Takase et al. 1981):

Joint coordinate programming consists of absolute axis rotation commands, instructing the robot to move each of its six axes to a defined rotation value.

A1 0, A2 10, A3 90, A4 20, A5 60, A6 25

Cartesian coordinate programming defines the location and orientation of the end-effector in a previously defined Cartesian coordinate system. This can either be *point-to-point* (PTP) commands where the end-effector moves from one position to the next with the least amount of axis rotation or linear (LIN) commands where the end-effector moves from one position to the next along a straight line.

X 10, Y 20, Z 40, A 45, B 75, C 15

3 Parametric Robot Programming for Architectural Design

Understanding the robot's data processing described in Section 2, it appears feasible to create KRL code via custom scripting (offline programming), thereby automating the fabrication process for architectural mass customization.

For most architectural purposes, the location of the tool has to be precisely controlled, therefore requiring the use of *Cartesian coordinate programming*. The kinematics of the robot do not necessarily have to be simulated in the parametric modeling environment, as they are solved according to the end-effector position/orientation in realtime when the KRL code is executed at the control unit. However, simulating the robot's kinematics in a design environment enables the designer to react to reachability-constraints or possible collisions, thereby optimizing the use of the available workspace.

3.1 OFFLINE PROGRAMMING VIA CUSTOMIZED SCRIPTING

A recent architectural example involving industrial robots and custom scripting is the *Pike Loop* project **(Figure 4)** by Gramazio and Kohler (Bärtschi et al. 2010), where a special "wiggled bond brick arrangement is mapped onto a generic double curved surface". This results in a CAD-created surface being processed by a script that deals with the various problems involved in robotic brick stacking, like the general arrangement, the boundary conditions and optimization issues **(Figure 5a)**. While the script is executed, the user cannot interact but has to wait until the script is finished; the user is then presented with the result and has to evaluate if the achieved output meets predefined requirements or if the result has to be changed. This can be done either by changing the predefined requirements, by modifying the initial CAD geometry, or by changing the code itself. Basically, the code acts as a *solver* for a sequence of clearly (pre)defined problems.

Figure 4. *Robotic fabrication for the Pike Loop Project by Gramazio and Kohler*

Even though the use of an industrial robot for bricklaying is a very innovative architectural application, the published digital workflow via customized scripting describes a typical architectural design process with a designer (this can be an architect) to create an aesthetic surface in CAD. A programmer (supposedly a computer scientist or mathematician) then solves the "bricklaying problem" by applying the geometric constraints of bricks to a predefined surface (similar to the known segmentation problem of any freeform surface in architecture), (Eigensatz et al. 2010) followed by a technician who postprocesses the geometric data output for the robot control data file (e.g. in a robot simulation program), similar to a contractor who has to postprocess received geometric data for his CNC-machines. The question arises here how to further customize the digital workflow to allow the user, i.e., the designer, to manipulate the initial CAD surface, the brick-layout and the robot control simultaneously?

3.2 A NEW APPROACH FOR ROBOT OFFLINE PROGRAMMING

The Pike Loop project shows how customized scripting allows architects to alter their designs parametrically in a linear process and how to execute parametric designs with an industrial robot set-up, in this case a robotic gripper. While the exemplary data flow described in Section 3.1 is suitable for many architectural applications, we argue that it is not ideal for a fluent *bottom-up architectural design to robot milling* workflow.

In addition to the desire of a fully parametric robot control, the functionality of common CAM software such as user-defined tool geometries (e.g. varying tool diameter or cutting lengths) within a single project, automated toolpath generation and collision checking has to be included in a parametric model for robot milling.

A well-integrated parametric design tool for robot milling and architectural design (e.g. for mass customization) requires the user (i.e., the designer) to be able to instantly and constantly interact with all parts of the code. Preparing data for robot milling is therefore a highly non-linear process.

In our previous research we have shown that a fully parametric design for robot milling greatly streamlines the manufacturing process as changes imposed by tool geometry, material properties or machine constraints are easy to implement. We have proposed several strategies to improve the workflow from digital design to production, such as including machine constraints in the parametric model or generating G-code with a Grasshopper definition. However, some workflow problems remained as the G-code still had to be post-processed before sending it to the robot's control unit. Recently, we demonstrated the use of a Grasshopper definition, capable of generating actual KRL code. However, maintenance of this complex cluster of Grasshopper components proved to be cumbersome, as there was no possibility of implementing a central graphical interface or intuitive design for mass customization. Solving these issues requires the development of an actual robot milling plug-in, as demonstrated by the KUKA|prc component in Section 4.1.

4 Designing a Parametric Component for Robot Milling

The main idea to facilitate working with robots is a *bottom-up design* where robot milling strategies determine the geometric design right from the beginning and therefore become an integral part of the design, always available in the background for analysis and simulation. It is possible to robotically fabricate infinite design variants of the current model at every time with just a few mouseclicks and without having to go through multiple export/import steps from CAD to CAM to the robot **(Figure 3)**. Collisions and insufficient tool length are detected at runtime, so that the design (i.e., the code) can be instantly revised.

Grasshopper was chosen as the parametric environment because of its modular, open structure and realtime preview. Especially important for robot milling is that Grasshopper's component system allows compartmentalizing solutions and reusing existing code, in order to adapt to emerging design requirements or problems. We refer to these functional groups of components as *modules* **(Figure 5b)**.

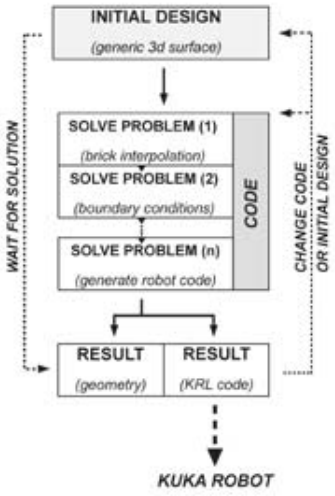

Fig. 5a

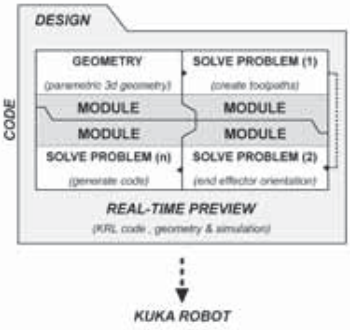

Fig. 5b

Figure 5a. *Dataflow comparison - linear versus non-linear processes: conventional scripting*

Figure 5a. *Integrated parametric approach for robot milling*

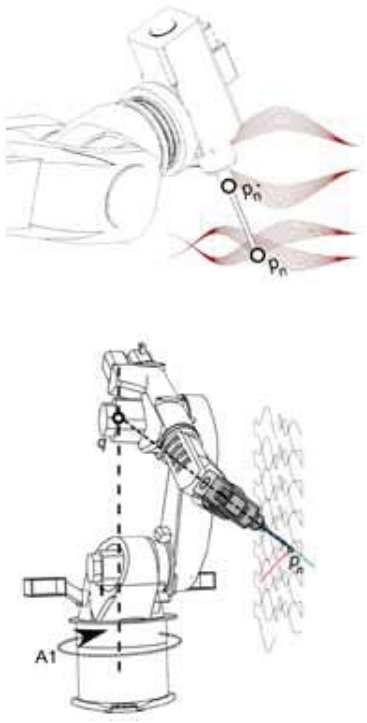

Fig. 6

Figure 6. *Necessary geometric input for KRL code generation: tool and guidecurve (left), orientation of the end-effector (right)*

For example, the designer can create more than only one geometric module and immediately generate toolpaths, simultaneously calculating the end-effector orientation, checking for collisions and creating the KRL code for each of them, just by sequentially plugging them into the toolpath creation module.

"Conventional" scripting can mostly be assumed to be solving clearly defined problems in a predefined sequence which also applies to most Grasshopper modules. In contrast to typical scripting code, however, the KUKA|prc component is defined as follows: by integrating robot constraints and important CAM functions such as toolpath generation into the parametric model, we generate a design tool for robot milling that instantly reacts to all changes, allows fluid and intuitive modifications in any of the modules, at any time. This offers the possibility of quickly evaluating designs for robot milling, simulating toolpaths and generating the machining code without having to deal with either hundreds of lines of code or several different CAD, CAM and kinematic software packages or postprocessing. The KRL component therefore establishes a feedback loop from CAAD to CAM to the robot and back again.

4.1 KUKA KRL-COMPONENT: GENERAL APPROACH

While designing the KUKA Robot Language component for Grasshopper we aimed to fulfill the following five requirements:

- The component has to integrate into the Grasshopper environment like a native component.
- It should focus on flexible, dynamic robot control without the feature-overload of commercial CAM software.
- Its calculations are to be completed in less than a second to retain the near-realtime feedback, with an additional quick preview mode.
- Its functionality should be transparent to the designer and customizable, wrapped in an accessible user interface.
- The output has to consist of human-readable, non-binary KRL code that can be executed directly at the robot, without any additional steps such as post-processing.

The resulting KUKA Robot Language Component for Grasshopper is written using the .NET framework (RhinoCommon SDK & Grasshopper SDK) in Visual Studio and seamlessly integrates with the default Grasshopper components. A project in Visual Studio can contain several classes (i.e., Grasshopper components) which compile into a single class library that can be easily distributed among users.

4.2 KUKA KRL COMPONENT: FUNCTIONALITY

We define the end-effector orientation and position along a toolpath in the KRL component with the help of three inputs **(Figure 6)**:

- The *tooltipcurve* defines the position of the tool center point (TCP) on the toolpath during a milling job at a point p_n.
- The *guidecurve* at p_n' defines the tool axis as the vector from p_n' to p_n.
- The *orientationpoint q* defines the rotation of the end-effector around the toolaxis and can be either a static point or a dynamic point along a curve. In **Figure 6**, *q* is placed along the robot's axis A1.

Synchronizing point p_n' and p_n is done via the curves' parameterization or by dividing the curves into an equal number of points. The resulting information consists of a point (XYZ) and Euler angles (ABC) which can be formatted and exported as KRL code.

Besides the parametric numeric (simulation slider) and geometrical input (toolpath points, collision geometry), the KRL component also provides a graphical user

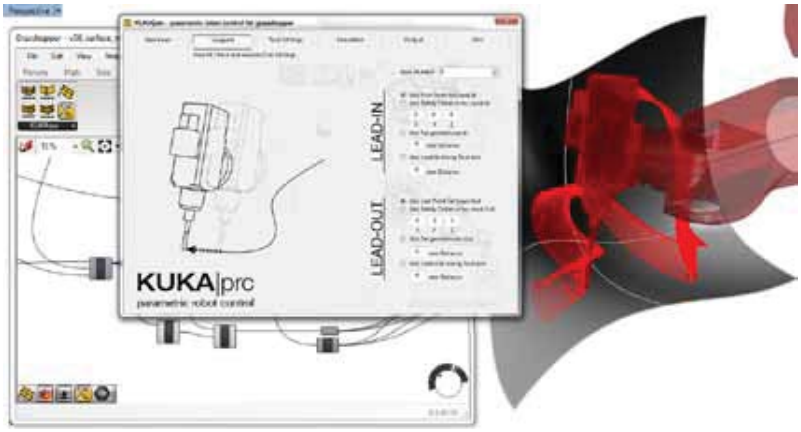

Fig. 7

interface, accessible within Grasshopper **(Figure 7)**. Simulated objects such as the robot's end-effector are generated parametrically within the component according to the tool parameters and do not require separate Grasshopper geometry input.

One of the main problems when using existing CAM software is that CAM software is a closed system that contains many functions from mechanical engineering, which are not necessarily required for architectural designs. The options available within the KRL component focus on the most essential features of CAM software:

- Save/load of configuration data
- Set up the robot's base including the referenced coordinate system
- Define the lead in and lead out (none/offset/tangential/along tool axis)
- Set tool number and properties (tool length/cutting length/diameter/feed rate)
- Collision control settings
- Visualization/viewport performance settings
- KRL file settings
- Enable/disable sequential file naming for mass customization

Whenever the parametric model is changed, the KRL files are simultaneously updated with the newest version in the directory defined by the output settings. For mass customization it is possible to override this setting and attach a sequence number to the filename so that separate files are automatically generated for each variation.

4.3 KUKA KRL COMPONENT: APPLIED DESIGNS AND EVALUATION

The KUKA KRL component's mass customization capabilities were tested with different parametric designs. One design task was to create a parametric EPS brick wall out of individual, stackable elements with varying openings. Its basic toolpaths consist of just two parametric curves that create a range of different physical, stackable 3D modules with varying geometry and openings **(Figure 8)**.

The Grasshopper definition in **Figure 8** allows various realtime previews which enable the designer to simultaneously follow the design and production process. The geometric generation of the parametric tooltip- and guidecurves (1), the individual modules to be produced (2), the stacked results (3), the automated toolpath generation (4) and a simulated end-effector are all shown at the same time in a single viewport. The end-effector simulation and KRL output (5) are handled by the new custom KRL component. This demonstrates our initial idea of using the parametric design environment as a custom design tool where changes to single *geometric*

Figure 7. *KUKA|prc - graphical interface for parametric robot control*

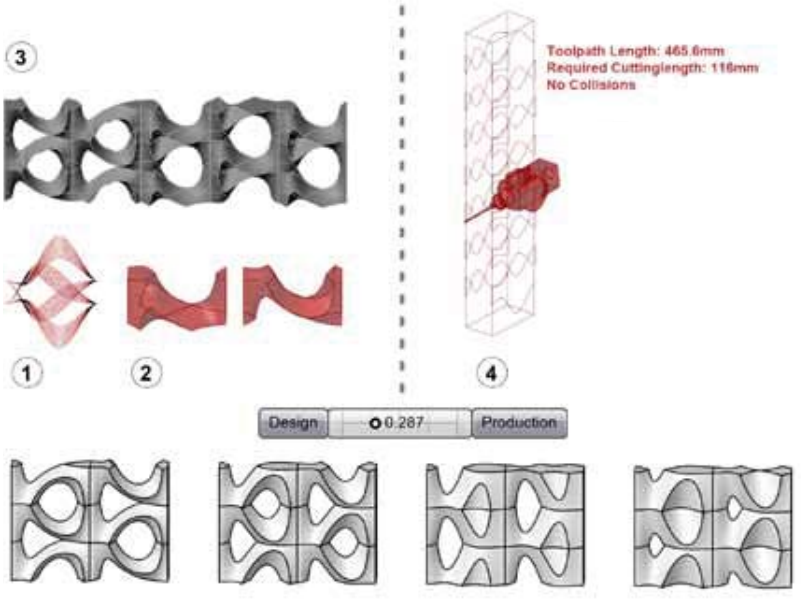

Fig. 8

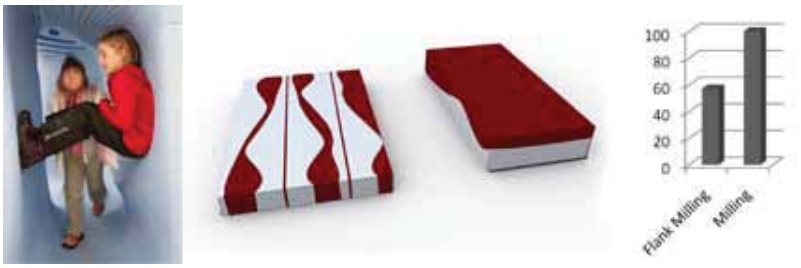

Fig. 9

modules, or to the overall design, have immediate consequences for the toolpath generation and vice versa.

In contrast to traditional CAM software, this modular approach allows an open and flexible parametric CAM environment, where the user does not necessarily have to rely on pre-set toolpath-strategies, but can substitute these components with his own parametric Grasshopper definitions.

To compare the Grasshopper integrated KUKA plugin with traditional CAD/CAM, two prototypes of the same parametric wall design were milled. The physical models of both results were indistinguishable, while the preparation with the commercial CAM software took only slightly longer compared to the KRL component. However, any successive variation would widen the time-gap as the CAD/CAM software has to be set up each time, while the KRL component can continue generating KRL code instantly. For the design variant in Figure 8, the toolpath consists of 1500 points. The KRL file is generated within 45ms on a 2GHz laptop, which would theoretically allow the automated output of 20 design variants per second. This rapid response allows designers to quickly output and fabricate design variants, without having to manually process each file in the CAM software and postprocess for the robot´s kinematics. A further advantage is that the calculations happen parallel to the design process: while the designer can "play" with the geometric entities such as curves and stackable 3D modules, the KUKA plugin provides feedback on fabrication feasibility and possible collisions.

In order to evaluate the Grasshopper-integrated KUKA plugin with a full scale architectural prototype, we revisited "Plug & Play," a robotically fabricated project from 2009/2010 that required the fabrication of 18 unique freeform panels for a 12m² freeform interior surface as part of a children´s playground design **(Figure 9)**. Different milling strategies such as roughening, finishing and flank milling were evaluated and successfully used with the KUKA plugin.

Figure 8. *Production immanent parametric model simultaneously showing realtime preview of geometry input (1), individual modules (2), final design (3), milling simulation with feasibility analysis (4) and KRL code (5)*

Figure 9. *12m² freeform surface segmented in 18 panels (left), comparing flank milling and point milling according to material efficiency (middle, right)*

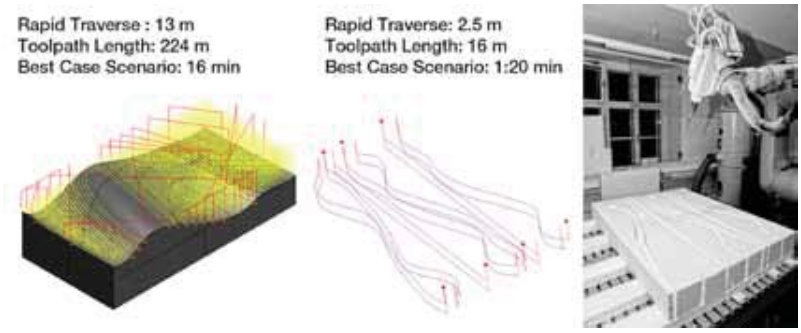

Fig. 10

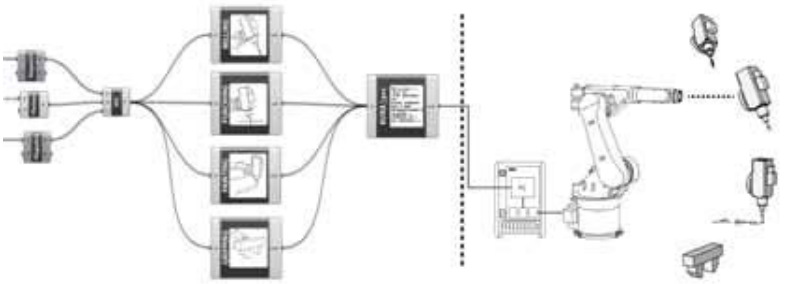

Fig. 11

Roughening and finishing are common point milling strategies that use the tooltip of the endmill to coarsely remove stock material layer by layer (roughening), and then to create the final surface finish (finishing). In contrast, flank milling (Schindler 2009) utilizes the whole cutting length of the endmill to tangentially cut finished surfaces out of the stockmodel.

Due to these inherent differences between point and flank milling strategies, two separate approaches were developed to produce the shape of the panels. For roughening/finishing, several blocks of XPS were used to mill the final surface. For flank milling, the freeform shape was approximated with ruled surfaces with the width of each surface corresponding to the height of a common XPS panel. A single cut was then performed to produce the finished surfaces.

Both from the computational and physical point of view, flank milling proved to be the best solution for this particular project. Compared to the point milling strategies, flank milling required approximately 80% less toolpath length, which equals machine time, and achieved more than 40% in material savings **(Figure 9)**. Similar results for flank milling applications were reported by Waldt (2005).

Figure 10 shows the comparison of the proposed milling strategies of an exemplary panel in commercial CAM software and the impact of flank milling on toolpath lengths and machine time.

The increased toolpath length of point milling also has consequences for the parametric design environment, as the robot code generation is noticeably slowed down by the excessive amount of data. In the freeform playground design only flank milling allows the designer to work in a fluent, near realtime design environment. The right choice of milling strategies is therefore crucial for the parametric design environment and efficiency of mass customization. Therefore the designer has to evaluate milling strategies before creating a parametric model. Any design-to-fabrication automation requires milling experience to allow such decisions.

The two realized examples, the parametric wall and the children´s playground design presented in this paper show that parametric design software with integrated CAD/CAM features is capable of accommodating customized and optimized fabrication strategies that would not be possible using traditional CAM software tools only. Significant savings in machining time and material costs, along with the ecological aspect of producing less waste, can be achieved by optimizing designs for fabrication instead of just sending the final 3D CAD file to the fabrication workshop. The KUKA component for Grasshopper enables a fluid workflow from optimized design to robotic fabrication.

Figure 10. *Freeform playground: comparing point- and flank milling strategies*

Figure 11. *Modular, parametric robot control using KUKA|prc*

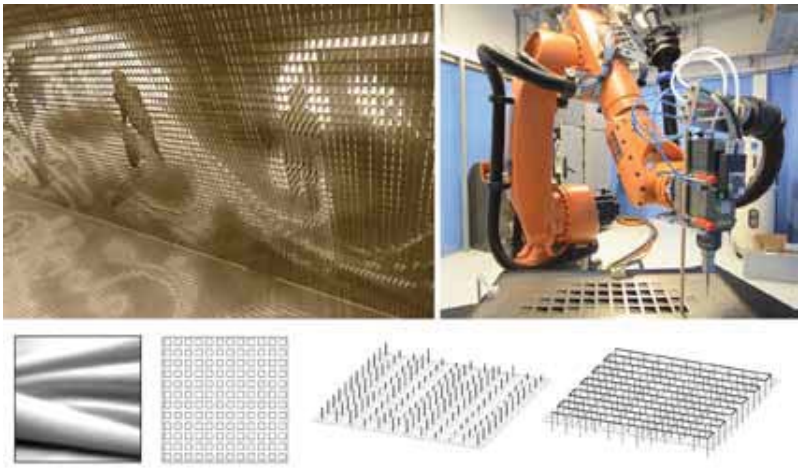

Fig. 12

Fig. 13

5 Conclusion and Outlook

Industrial robots have been applied in full-scale construction – as a replacement for manual labor – in Japan since the 1980s with little success due to automation difficulties and economic reasons. As Bechthold (2010) states, the development of automated programming strategies is fundamental for using robots in "customized construction".

Our intent in research is to enable a wide range of architects in the use of robots with a similar ease of use compared to conventional CNC-machines and an intuitive approach towards mass customization. The presented parametric interface in this paper is a first step for industrial robots to be used as full scale construction technology by a broad range of designers and CAD users.

From the architectural design point of view we would like to "play" with *robotically manufactured designs*, where all necessary constraints are integrated in a parametric model. Our approach of designing the KUKA|prc component for milling, is - compared to Gramazio and Kohler´s *robotically bricklaid walls* - of course a geometrically less complex problem to solve, but demonstrates a certain playfulness in design and ease in the creation of an interface for the use of a technically sophisticated CNC machine such as a KUKA robot. The multifunctionality of Grasshopper can be compared to the multifunctionality of an industrial robot where the designer can *plug* and *unplug* different tools like milling-cutters for robot milling or gripping tools for bricklaying **(Figure 11)**. It allows the expectation that the presented KUKA|prc component in this paper is modifiable to achieve a wide variety of robot applications by simple *plugging* and *unplugging* different geometric solutions or tool constraints, enabling us to integrate industrial robots in CAAD and move beyond CAD/CAM as we know it **(Figures 12, 13)**.

Acknowledgements

The projects in **Figure 12** (Parametric Punching by Michael Vasku) and **Figure 13** (Lightpainting by Vera Kumer) resulted from testing the KUKA|prc plugin in the lecture "Digital Design to Production" at Vienna UT. We would like to thank all of our students for their enthusiasm, and the Association for Robots in Architecture, Vienna University of Technology, the Innovative Project "Geometric Model Building with CNC-Technologies", the "7-Axis CNC Milling with an Industrial Robot in Architecture" project, and KUKA CEE for their kind support of our research.

Figure 12. *Robotic fabrication beyond CAD/CAM: parametric punching – toolpath generation via image sampling*

Figure 13. *Robotic fabrication beyond CAD/CAM: lightpainting*

References

Bärtschi, R., M. Knauss, T. Bonwetsch, F. Gramazio, and M. Kohler. 2010. Wiggled Brick Bond, *Advances in Architectural Geometry* 2010: 137-148, Vienna: Springer.

Bechthold, M. 2010. The Return of the Future- A Second Go at Robotic Construction , *Architectural Design* 80(4):116-121. Hoboken: Wiley and Sons.

Biggs, G. and B. MacDonald. 2003. A Survey of Robot Programming Systems. *Proceedings of the Australasian Conference on Robotics and Automation*, *CSIRO*. Brisbane.

Brell-Cokcan, S. and J. Braumann. 2010. A New Parametric Design Tool for Robot Milling. *Proceedings of the 30th Annual Conference of the Association for Computer Aided Design in Architecture*: 357-363. New York.

Brell-Cokcan, S., M. Reis, H. Schmiedhofer, and J. Braumann. 2009. Digital Design to Digital Production: Flank Milling with a 7-Axis Robot and Parametric Design. *Computation: The New Realm of Architectural Design. 27th eCAADe Conference Proceedings*: 323-330. Istanbul.

Eastman, C., P. Teicholz, R. Sacks, and K. Liston. 2008. *BIM Handbook*. Hoboken, New Jersey: Wiley and Sons.

Eigensatz, M., M. Kilian, A. Schiftner, N. Mitra, H. Pottmann, and M. Pauly. 2010. Paneling Architectural Freeform Surface. *ACM Transactions on Graphics* 29(4). New York: ACM.

Gramazio, F. and M. Kohler. 2008. *Digital Materiality in Architecture*. Baden: Lars Müller Publishers.

Iwamoto, L. 2009. Digital Fabrication: *Architectural and Material Techniques*. New York: Princeton Architectural Press.

Kolarevic, B. and K. Klinger. 2008. *Manufacturing Material Effects – Rethinking Design and Making in Architecture*. New York: Routledge.

KUKA Robotics. 2003. *KR C2 / KR C3 Expert Programming*. Augsburg: KUKA Roboter GmbH.

Mühe, H., A. Angerer, A. Hoffmann, and W. Reif. 2010. On reverse-engineering the KUKA Robot Language. *Proceedings of the DSLRob'10, IEEE/RSJ International Conference on Intelligent Robots and Systems 2010*, Taipeh.

Piller, F. 2004. Mass Customization: Reflections on the State of the Concept. *The International Journal of Flexible Manufacturing Systems* 16:313-334. Springer.

Ramani, K., R. Cunningham, S. Devanthan, J. Subramaniam, and H. Patwardhan. 2004. *Technology Review of Mass Customization*. Product Models Organization Conference 2004.

Scheurer, F. 2010. Materialising Complexity. *Architectural Design* 80(4). Hoboken: Wiley and Sons.

Schindler, C. 2009. Flankenfräsen. In *Ein architektonisches Parametrisierungsmodell anhand Fertigungstechnischer Kriterien, dargestellt am Holzbau*, Zürich: ETH Zürich.

Takase, K., R. Paul, and E. Berg. 1981. A Structured Approach to Robot Programming and Teaching, *IEEE Transactions on Systems, Man, and Cybernetics* 1(4). New York: IEEE.

Waldt, N. 2005. *NC-Programmierung für das fünfachsige Flankenfräsen von Freiformflächen*. Hannover: Universität Hannover.