Visual Programming for Interactive Robotic Fabrication Processes

Process flow definition in robotic fabrication

Johannes Braumann[1], Emanuel Gollob[2], Karl Singline[3]
[1,2,3]Creative Robotics, UfG Linz, Austria
[1]johannes@robotsinarchitecture.org [2]emanuel@emanuelgollob.com
[3]karl.singline@ufg.at

Visual, flow-based programming environments in architecture and design are built to control data flow but not process flow. However, controlling the process flow is essential for interacting with robotic fabrication processes, so that they can react to input such as user interaction or sensor data. In this research, we combine two visual programming environments, utilizing Grasshopper for defining complex, robotic toolpaths, and Unity Visual Scripting for controlling the overall process flow and process interaction. Through that, we want to enable architects and designers to define more complex, interactive production processes, with accessible, bespoke user-interfaces allowing non-experts to operate these processes - a crucial step for the commercialization of innovations. This approach is evaluated in a case study that creates a mobile, urban microfactory that prototypically fabricates location-specific objects through additive manufacturing.

Keywords: *Visual Programming; State Machines; Industrial Robotics; Unity Visual Scripting.*

INTRODUCTION

A priority of our research into parametric robot control (Braumann and Brell-Cokcan, 2011) has always been to increase the accessibility for new user groups, targeting primarily the creative industries with their existing expertise in visual programming.

Visual programming environments such as McNeel Grasshopper and Autodesk Dynamo, supported by specialized plugins, today allow the definition of geometrically complex, individualizable fabrication processes that go far beyond the scope of industry-standard CAM (Computer-Aided Manufacturing) software, but due to their inherently acyclic design are significantly limited when it comes to the handling of user interaction, real-time data and decision-making.

Especially when robotic processes move from academic research to commercial application, there is a need to create maintainable systems with easy user-interaction that allow non-experts to manage a production process.

An area where accessibility, low-code, and decision-making intersect is game development. Popular game engines like Unity and Unreal have created rich ecosystems that support developers with their visual programming environments.

This paper presents a prototypical system that allows accessible visual programming of interactive robotic fabrication processes within Unity. By building upon established user interaction concepts and programming environments, we aim to democratize the definition of complex, interactive

robotic fabrication sequences. As such, the research targets designers who are already familiar with parametric visual programming environments and get empowered to create flexible, branching fabrication logic informed by user-input or sensor data.

In a commercial context, this logic can then be "packaged" so that non-experts only need to interact with a focused, custom-tailored UI that guides them through a fabrication process, without requiring expertise in the underlying visual programming code.

This approach is verified through the rapid prototyping of a mobile, robotic 3D printing platform that acts as an urban microfactory, enabling local and customized fabrication while operated by a non-coder.

FLOW-BASED PROGRAMMING FOR ARCHITECTURAL ROBOTICS

Industrial robots are today commonly encountered in architectural research, education, and increasingly also in practice.

Figure 1
Real-time robotic process defined through flow-based programming, using Grasshopper

An enabling factor in that regard has been the development of accessible robot simulation and programming tools like HAL (Schwartz, 2013), KUKA|prc (Braumann and Brell-Cokcan, 2011) (see figure 1) and the open-source Robots (Soler, 2021).

Due to these and other interfaces, sending and receiving data from a robot has been realized in several architectural applications, for example by Amtsberg et al (2015) and Johns and Anderson (2018). Frameworks for real-time robotic control have been presented by Braumann and Brell-Cokcan (2015) and Garcia del Castillo y López (2019).

The underlying programming paradigm of flow-based programming is relatively old, having been developed by Morrison (2010) in the late 60s for IBM, building upon the concept of co-routines by Conway (1963). Today, flow-based visual programming environments like McNeel Grasshopper can be considered part of the trend towards low-code (Fryling, 2019), allowing non-experts to create geometric algorithms intuitively.

Of particular benefit to non-programmers is Grasshopper's immediate reaction to user interaction, coupled with a well-thought-out approach to user interaction through colouring and visual cues, and its thorough catching of most exceptions.

As Grasshopper enforces an acyclic dataflow, current workflows for robot programming in Grasshopper work particularly well for processes that are self-contained and do not depend on previous iterations, like for example typical production processes where some parameters are set, and the resulting robot trajectory is then either written into a file or directly streamed to the robot.

However, within the scope of interactive robotic fabrication, we see a major challenge in the fact that Grasshopper's data flows cannot be natively directed and looped.

As such, users must either use third-party plugins (Zheng, Zhe, and Yang, 2019) to implement ways of feeding data back to the start of a Grasshopper definition (Celani and Vaz, 2012) or switch to systems like Processing for generative processes (Pantazis, 2017; Elashry and Glynn, 2014).

In reference to human's tendency to adjust ideas in the process of making, Kim et al. (2017) outlined a speculative system for interactive human-machine fabrication that would allow a bidirectional fabrication process, seeing machines as

collaborators instead of output devices; such could currently not be realized natively on Grasshopper.

We therefore see a need for systems that allow a more accessible and visual definition of complex data flows, to enable interactive fabrication, where the data may originate from a user through a graphical user interface, a machine learning model, an external sensor system, or another robot.

METHODS: TOWARDS INTERACTING WITH FABRICATION

In the wider field of robotics, "pre-defined" processes - as in the fabrication of customized, architectural components - are comparably rare. In contrast, dynamic processes that adapt to navigate in new environments (Kruse et al, 2013), grasp unknown objects (Zeng et al, 2019) and in general react to real-time sensor data are the norm. Popular software environments like ROS (Quigley et al, 2009) provide a middleware but generally depend on text-based programming languages like Python and C++.

Reacting to user input, dealing with multiple object states, and making decisions based on various input data are core aspects of game development that are therefore well represented in these environments. Within the context of this research, Unity Visual Scripting (UVS) was chosen as both Unity, and Grasshopper support plugins written in the .NET framework, allowing us to make the existing capabilities of KUKA|prc available in both environments, greatly facilitating the exchange of data, as well as troubleshooting.

There are already several projects within the field of architecture and robotics that were built on Unity, e.g., for creating a graphical user interface for complex robotic 3D printing (Soler, 2017) and acting as a machine learning environment (Hosmer, 2019).

Bringing KUKA|prc to UVS

While initially a Grasshopper plugin, KUKA|prc is written as a .NET Standard library that can be used on all platforms that support the .NET framework.

Integrating the CAD-originating KUKA|prc into the computer-graphics-based Unity requires us to translate the default right-handed coordinate system (Z+ up) to Unity's left-handed (Z+ forward) coordinate system and to express distances in meters instead of millimetres.

A significant difference compared to Grasshopper is that UVS also provides a graphical way to define the data flow (see figure 2), so that e.g., after an if-else node, only the nodes connected to the chosen option are executed. Furthermore, multiple so-called flow-graphs (in Grasshopper: definitions) can be executed in parallel.

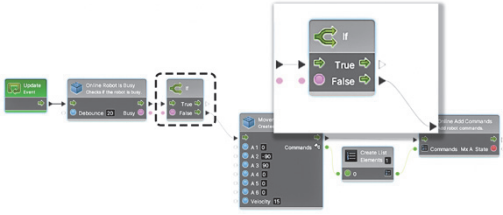


Figure 2
UVS allowing process flow control in addition to data flow

This is unlike Grasshopper, where only one definition can be active at a time. In addition, UVS also introduces state-graphs, which are state machines that can be used to define the transition of one flow-graph to another one.

The concept of a state machine dates back to an even earlier date than flow-based programming (McCulloch and Pitts 1943). UVS uses a Mealy-style finite state (Mealy, 1955) consisting of a finite number of states that are connected through transitions, which in turn are tied to certain conditions.

It can therefore be easily represented by a state transition diagram, which in UVS's state-graph not only visualizes the data flow but becomes a visual way of defining and connecting program states. A designer can therefore sketch out the process flow as a state graph and then define the individual flow graphs within the state machine.

Within the context of architecture, Jensen and Das (2020) propose the use of state machines for human-robot collaboration, utilizing the addon Metahopper with custom Python scripts to realize a state machine within Grasshopper. While not

referred to as a state machine, the "feedback loop" by Moorman et al (2016) fulfils a similar purpose. Garcial (2019) utilizes a state-based "enactive robot system" that can communicate with external visual programming environments.

CAD in Unity

In UVS, the user creates a state machine ("state graph") and then adds various flow states, each of which contains a script-graph, with the transitions also represented by a flow-graph. Once a data flow reaches a transition element, the state machines change to the according state and executes the associated flow, until the next transition is triggered. This allows the user to define complex interactive fabrication processes through a flow-based programming tool.

However, the generation of robot trajectories is challenging, as Unity is missing many of Rhinoceros 3D's and Grasshopper's geometric features.

We therefore use Rhino Compute to allow access to Rhinoceros 3D functions within Unity through a representational state transfer (REST) application programming interface.

Doing so allows us to split the relatively light user-interaction and data-flow tasks from the more computation heavy path planning and toolpath verification. As such, even a relatively low-powered device like a tablet or mixed reality headset can orchestrate a complex robotic production process.

CASE STUDY: WANDERING FACTORY

In order to verify the capabilities of the developed system, we have applied it towards creating an interactive, mobile, robotic fabrication process that…

- …allows UI-based operation of a production process by an unskilled user.
- …may be programmed by a non-coder solely through visual, flow-based programming in Grasshopper and UVS.

The Wandering Factory is intended as a demonstrator within the context of the FabCity Global Initiative, towards enabling the return of manufacturing to cities. The demonstrator (see figure 3) consists of four dedicated parts that are linked through the software architecture that was developed as part of our research: A KUKA Agilus KR10-R1100-2 robotic arm, a Mattro ROVO-2 tracked, electric vehicle, an Intel RealSense 3D camera and a Noztek pellet extruder. Communication is established through a variety of interfaces: KUKA mxAutomation controls the robot in real-time, the vehicle uses CAN bus for communication, the 3D camera is connected via USB and the extruder via the EtherCAT fieldbus with a Beckhoff-PLC (programmable logic controller) in between. These data sources are managed on a compact PC running Unity which at the same time provides a 3D visualization and a user interface.

The idea of the demonstrator is to showcase approaches towards local, customized, and sustainable fabrication: An environment is scanned, and the resulting point cloud used to customize an object so that it precisely fits at the captured location. Toolpaths are automatically generated so that the object may be immediately 3D-printed by the industrial robot using recycled materials, without requiring an expert-user to be on-site

Figure 3
Wandering Factory demonstrator (left), individualized, locally adapted 3D print representing urban furniture (right).

Implementation

The Wandering Factory is implemented through two UVS graphs: A script graph is constantly running in

Figure 4
State-graph representing the Wandering Factory project in UVS

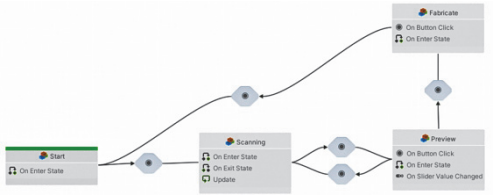


Figure 5
State-graph representing the Wandering Factory project conceptually

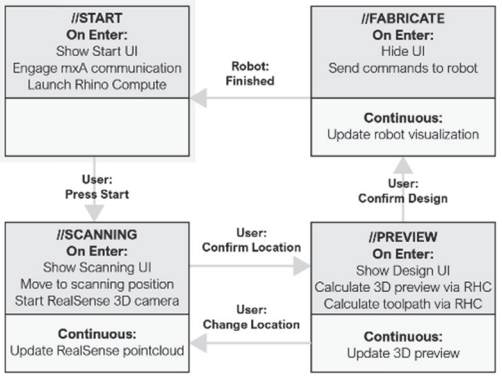


Figure 6
Point cloud visualization in the user interface

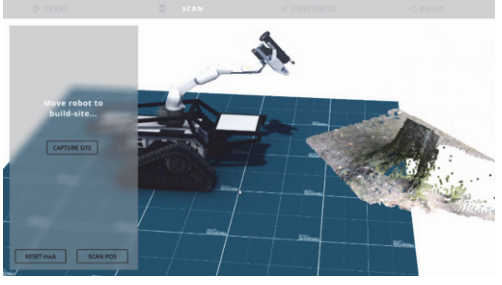
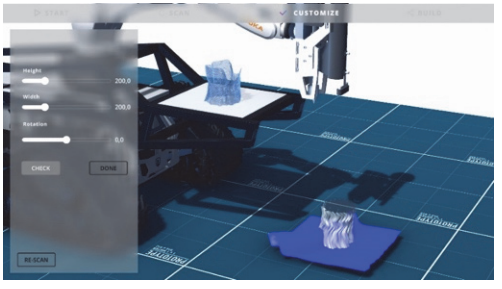


Figure 7
3D print preview based on scan data, generated via Rhino Compute



the background and manages the communication with the KUKA robot via mxAutomation, while a state graph largely follows the user's interaction with the user interface, processing and visualizing the data coming from the robot as well as the various sensors and connected systems.

The unskilled user is guided through the fabrication process in five steps that correspond to the interface sections (see figure 4 and 5).

Initially, the user moves the mobile platform to the desired location. In the "Scanning" state, the point-cloud generated by an Intel Realsense 3D camera is displayed in real-time within the robot's workspace (see figure 6). Once on-site, the "Preview" state saves the point cloud to a local file as embedding it into the Grasshopper file would lead to a decrease in performance. The provided Grasshopper script then turns the point cloud into a NURBS surfaces, fits the targeted print object onto the surface and generates the necessary non-planar toolpaths to fabricate it with the robot's pellet extruder. Finally, the script returns both the list of serialized robot commands as well as the serialized preview mesh geometry, which is then displayed in the Unity viewport (see figure 7). The user can then either change some core parameters of the targeted print object, or proceed to the "Fabricate" state, which deserializes the KUKA|prc robot movement commands and streams them to the robot via the mxAutomation interface, while visualizing the robot's current position in the user interface. Once completed, the user can immediately start a new process.

Evaluation

One of the most visible projects towards mobile fabrication in an architectural context is ETH's "In-Situ Fabricator" (Buchli et al, 2018). While its capabilities are clearly far beyond the presented case study, doing so required a team of many researchers and a large budget over multiple years, creating a multi-layered, custom software architecture building upon software such as OpenCV, ROS, and Eigen as well many custom-developed tools

In contrast, the Wandering Factory was put together within 6 weeks, with a minimal budget and using a control system that is built exclusively on flow-based programming, showcasing the increasing accessibility of technologies that were previously limited to high-end research institutions.

RESULTS AND DISCUSSION

We consider the combination of using UVS for the global process planning and user interaction with Grasshopper through Rhino Compute to be a powerful way for defining complex robotic fabrication processes in a visual, low-code way. That approach builds upon the designer's existing knowledge with Grasshopper but expands its scope significantly towards enabling processes that require decision making based on human or machinic input - from simply waiting for user interaction to more complex robot behaviour defined through the state machine.

While the underlying technologies like state machines and flow-based programming belong to the foundations of computation, the accessibility of both Rhinoceros 3D and Unity sets the present approach apart from other solutions.

Both software environments benefit from their diverse, user-focused ecosystems that greatly expand their core-functionality beyond their initially intended purpose.

We see this prototypical case study as a proof of concept and a first step towards making visual programming of complex interactive robotic fabrication more accessible to experts and designers with a background in parametric design.

Outlook

For the future, we see two connected areas where further development of the presented technologies will potentially lead to innovation: The immediate interaction with machinic processes, as well as the easier integration of advanced computational methods and technologies, like mixed reality and machine learning.

The presented case study uses the transition of states mostly in synchronization with the user interface, following the user input. When working with real-time sensor data, these cycle times may be reduced towards enabling fully interactive fabrication processes that immediately react to user input, changes in the robot's environment or the behaviour of complex materials. Peng et al. (2018) presented approaches towards interactive fabrication for their Robotic Modelling Assistant.

Relating to mixed reality, Braumann and Singline (2021) presented a system for linking a WiFi-connected HoloLens 2 mixed reality headset through a robust, latency-tolerant web technology to mxAutomation. This approach would fit the presented technology, keeping the process control in the headset but outsourcing low-latency communication and intensive computing tasks to an external PC.

Through the Unity Machine Learning Agents Toolkit (Juliani et al, 2018), Unity has exposed machine learning processes in a very accessible way, with a specific focus on applying it in a 3D environment. Besides being used to train agents for gaming environments, recent research has also shown its potential for training robotic agents in the context of architecture (Hosmer, 2019).

Compared to Grasshopper, where machine learning plugins utilize Q-learning through the Accord.Net library (Tamke, Nicholas and Zwierzycki, 2018), ML Agents provides a wider variety of reinforcement learning techniques, model-based training as well as with its grid sensors ready to use tools for complex robotic path training in unstructured environments.

Looking ahead, we expect the Unity ML Agents Toolkit to be programmable from within UVS in the near term. In combination with the technology presented in this paper, this integration will make deep reinforcement learning with industrial robots accessible to non-coders.

We hope that the research presented in this paper will contribute towards enabling new, interactive robotic processes that so far were limited

by the constraints of popular visual programming environments, while also making it possible to integrate innovative tools and interfaces like machine learning and mixed reality in the near future.

REFERENCES

Amtsberg, F., Raspall F., and Trummer A. (2015). "Digital-Material Feedback in Architectural Design." In Proceedings of the 20th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA 2015).

Brandt, E., & Dannenberg, R. B. (1998). "Low-Latency Music Software Using off-the-Shelf Operating Systems." In Proceedings of the International Computer Music Conference. San Francisco.

Braumann, J., & Brell-Cokcan, S. (2011). "Parametric Robot Control: Integrated CAD/CAM for Architectural Design." In Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture, 242–51. Banff.

Braumann, J., & Brell-Cokcan, S. (2015). "Adaptive Robot Control - New Parametric Workflows Directly from Design to KUKA Robots." In Real Time - Proceedings of the 33rd ECAADe Conference, 243–50.

Braumann, J., & Singline, K. (2021). "Towards Real-Time Interaction with Industrial Robots in the Creative Industrie." In 2021 IEEE International Conference on Robotics and Automation - ICRA.

Brell-Cokcan, S., & Braumann J. (2015). "Toward Adaptive Robot Control Strategies." In ACADIA 2105: Computational Ecologies: Design in the Anthropocene, 223–31. Cincinnatti.

Buchli, J., M. Giftthaler, N. Kumar, M. Lussi, T. Sandy, K. Dörfler, and N. Hack. 2018. "Digital in Situ Fabrication - Challenges and Opportunities for Robotic in Situ Fabrication in Architecture, Construction, and Beyond." Cement and Concrete Research, SI : Digital concrete 2018, 112 (October): 66–75. https://doi.org/10.1016/j.cemconres.2018.05.013.

Celani, G., & Vaz, C. E. V. (2012). "CAD Scripting and Visual Programming Languages for Implementing Computational Design Concepts: A Comparison from a Pedagogical Point of View." International Journal of Architectural Computing 10 (1): 121–37. https://doi.org/10.1260/1478-0771.10.1.121

Conway, M. E. (1963). "Design of a Separable Transition-Diagram Compiler." Communications of the ACM 6 (7): 396–408. Available at: https://doi.org/10.1145/366663.366704

Davis, D., Burry, J., & Burry, M. (2011). "Understanding Visual Scripts: Improving Collaboration through Modular Programming." International Journal of Architectural Computing 9 (4). https://doi.org/10.1260/1478-0771.9.4.361

Elashry, K., & Glynn, R. (2014). "An Approach to Automated Construction Using Adaptive Programing." In Robotic Fabrication in Architecture, Art and Design 2014, edited by Wes McGee and Monica Ponce de Leon, 51–66. Cham: Springer International Publishing. Available at: https://doi.org/10.1007/978-3-319-04663-1_4 (Accessed: 27 March 2022)

Fryling, M. (2019). "Low Code App Development." Journal of Computing Sciences in Colleges 34 (6): 119.

Garcia del Castillo y López, J. L. (2019). "Robot Ex Machina." In Proceedings of the 39th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA).

Hosmer, T., & Tigas, P. (2019). "Deep Reinforcement Learning for Autonomous Robotic Tensegrity (ART)." In Proceedings of the 39th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA).

Jensen, M. B., & Das, A. (2020). "Technologies and Techniques for Collaborative Robotics in Architecture - - Establishing a Framework for Human-Robotic Design Exploration." In Proceedings of the 25th CAADRIA Conference.

Johns, R. L., & Anderson, J. (2018). "Interfaces for Adaptive Assembly." In Proceedings of the 38th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA).

Juliani, A., Berges, V. P., Vckay, E., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2018). "Unity: A General Platform for Intelligent Agents." ArXiv:1809.02627 [Cs, Stat].

Kim, J., H. Takahashi, H. Miyashita, M. Annett, and T. Yeh. 2017. "Machines as Co-Designers: A Fiction on the Future of Human-Fabrication Machine Interaction." In Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems, 790–805. CHI EA '17. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3027063.3052763.

Kruse, T., Pandey, A. K., Alami, R., & Kirsch, A. (2013). "Human-Aware Robot Navigation: A Survey." Robotics and Autonomous Systems 61 (12): 1726–43. https://doi.org/10.1016/j.robot.2013.05.007

Mealy, G. H. (1955). "A Method for Synthesizing Sequential Circuits." The Bell System Technical Journal 34 (5): 1045–79. https://doi.org/10.1002/j.1538-7305.1955.tb03788.x

Moorman, A., Liu, J., & Sabin, J. E. (2016). "RoboSense: Context-Dependent Robotic Design Protocols and Tools." In Proceedings of the 36th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA).

Morrison, J. P. (2010). Flow-Based Programming: A New Approach to Application Development. J.P. Morrison Enterprises.

Pantazis, E., & Gerber, D. (2017). "Emergent Order through Swarm Fluctuations - A Framework for Exploring Self-Organizing Structures Using Swarm Robotics." In Proceedings of the 35th ECAADe Conference.

Peng, H., J. Briggs, C.Y. Wang, K. Guo, J. Kider, S. Mueller, P. Baudisch, and F. Guimbretière. 2018. "RoMA: Interactive Fabrication with Augmented Reality and a Robotic 3D Printer." In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, 1–12. CHI '18. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3173574.3174153.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009). "ROS: An Open-Source Robot Operating System." In Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics. Kobe, Japan.

Schwartz, T. (2013). "HAL." In ROB|ARCH, edited by Sigrid Brell-Çokcan and Johannes Braumann, 92–101. Vienna: Springer. https://doi.org/10.1007/978-3-7091-1465-0_8

Soler. (2021). "Visose/Robots." [Online]. Available at: https://github.com/visose/Robots (Accessed: 27 March 2022)

Soler, V., Retsin, G., & Jimenez Garcia, M. (2017). "A Generalized Approach to Non-Layered Fused Filament Fabrication." In Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA).

Tamke, M., Nicholas, P., & Zwierzycki, M. (2018). "Machine Learning for Architectural Design: Practices and Infrastructure." International Journal of Architectural Computing 16 (2): 123–43. https://doi.org/10.1177/1478077118778580

Zeng, A., Song, S., Yu, K. T., Donlon, E., Hogan, F. R., Bauza, M., ... & Rodriguez, A. (2018) "Robotic Pick-and-Place of Novel Objects in Clutter with Multi-Affordance Grasping and Cross-Domain Image Matching." The International Journal of Robotics Research,

Zheng, H., Guo, Z., & Liang, Y. (2019). "Iterative Pattern Design via Decodes Python Scripts in Grasshopper." In Proceedings of the 18th CAAD Futures Conference.