Journal of
Professional Communication jpc

Live robot programming

Jason Lim★

Singapore-ETH Centre for Global Sustainability (Singapore)

Article Info

Article Type:
Research Article

Article History:
Received: *2013-08-12*
Revised: *2014-04-14*
Accepted: *2014-04-18*

Keywords:
Live programming
Max/MSP
Robotics

Abstract

Industrial robots are general-purpose machines. To perform a specific task, a robot is given instructions. This process is referred to as robot programming and can be approached in several ways. In online/teach programming, an operator controls the robot using a teach pendant and leads it through its task. This is recorded and can be replayed. In offline programming, instructions are pre-written in a programming language on a separate computer and then sent to the robot for execution. Referencing the field of computer music, this paper explores an alternative approach – live programming. Here, instructions are written and edited on-the-fly, triggering immediate robotic responses. As a proof of concept, the music synthesis software Max/MSP is extended with robot programming functionalities and used as a live environment to control a robotic folding process. This paper describes the development of a Max/MSP patch for folding aluminum strips and discusses the results of its use.

The industrial robot arm is reminiscent of a human arm. It has a shoulder, an elbow and a wrist; it reaches out to touch and manipulate objects. However, in place of a hand, the robot has a tool called an end-effector, which is replaceable. While humans control their arms innately, a robot must be instructed on how to move and what to do. By changing the end-effector and instruction set, a robot's physical capabilities and behavior can be modified. This makes it a highly versatile machine. Though originally developed for use in factories, industrial robots have found application in other domains, including creative practices such as art (Diaz, 2012), music (Bökesoy & Adler, 2011) and architecture (Gramazio & Kohler, 2008).

★*Corresponding author (Jason Lim)*
Email: jason.lim@arch.ethz.ch Phone: + 65-84814300

Programming may be understood as a means to communicate a robot user's intention to the machine. The two standard robot programming approaches are online and offline. A third approach, live programming, is proposed as an alternative. Conceptually, the three can be likened to different pedagogical methods – ways of teaching a robot what it needs to do. An online program is a demonstration; an offline program resembles an instructional manual; while a live program aspires to be conversational.

A case study is set up to explore the potentials of the live programming approach. The music synthesis software Max/MSP is extended with custom robot programming functionalities and used as a live environment to control a robotic folding process. A Max/MSP program (patch) is written that provides a lightweight robotic arm with real-time instructions on how to fold thin aluminium strips. The objective is to produce a variety of folded forms by running the patch or editing it on-the-fly. The development and use of the patch is described and the merits and drawbacks of live programming as a means of human-robot interaction is discussed.

Online programming – teaching by demonstration

Online programming encompasses *teach pendant* and *lead-through programming*. A human operator runs a robot through an entire process by controlling it via the teach pendant, a handheld control terminal, or by manually leading it. The sequence of motions and actions performed in this demonstration is recorded as a program that can be replayed.

Online programming is intuitive because it involves the direct manipulation of the robot. Thus, one of its key advantages is accessibility. It is an effective approach for programming processes that require repetition. However, a program created in this way is highly specific. If the robotic process changes, the program cannot be modified and reused. For example, to produce a range of unique folded forms, the robot has to be retaught from scratch each time. In this case, online programming would be too time-consuming to be practical.

Offline programming – writing an instruction manual

In offline programming, instructions are created separately on a computer. They have to be written in a programming language which the robot can interpret. Robot manufacturers provide such languages which are specific to their machines. A program is written, saved and uploaded to the robot, which then either completes the task or fails due to an unanticipated contingency. Offline programming may be likened to writing a manual. The task is known in advance, instructions are pre-written and once published/uploaded, cannot be changed.

One disadvantage of offline programming is its inaccessibility. Users have to learn the syntax and semantics of a robot programming language before being able to use it. Furthermore, these languages are often inexpressive and use technical notations; this raises the difficulty of reading and writing such programs. However, complex logics are easier to express in programming languages as they offer constructs such as conditionals and loops for structuring code. Furthermore, an offline program may be generalized. For example, by accepting an angle argument, a single program can produce different folds.

Live programming – having a conversation

In conventional programming practices, code is written or edited, then compiled into a program and subsequently run. There is a significant delay between a programming action and produced results. Live programming eliminates compilation and combines the two remaining steps into one. Code changes yield immediate feedback.

In a live robot programming scenario, the machine is in a continuous state of attention and reacts instantly to new instructions. These instructions may be notated in textual or graphical programming languages. Such a program may therefore be generalized, while a program created through online/teach methods is specific. In comparison to offline programming, the instruction set need not be pre-planned in its entirety. A programmer may give an instruction, observe the robot's reaction and then decide on the next instruction. One responds to the other in a conversational manner.

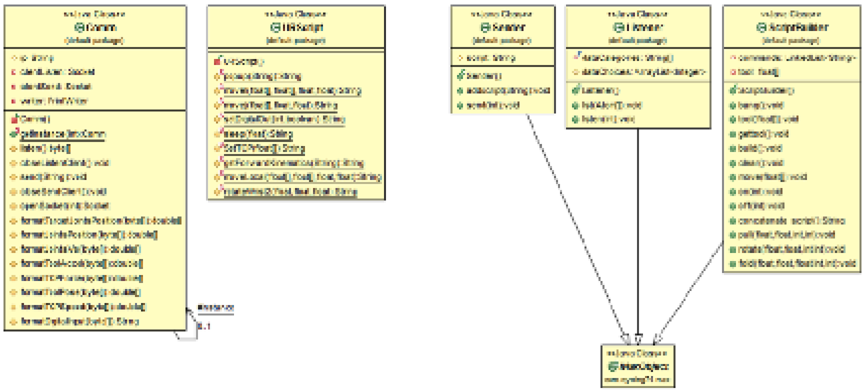
Extending Max/MSP for live robot programming

Live programming is widely practiced in the field of computer music. Musicians want to utilize algorithmic procedures for composition and be able to edit aural results in real-time. Many audio synthesis software systems today are thus live programming environments. Text-based examples include SuperCollider and ChucK, while graphical based examples include PureData and Max/MSP.

Max/MSP is selected as an environment to test the application of live programming concepts to robotics. A Max/MSP program, also referred to as a patch, is a data flow graph with objects as the nodes and patch-cords as the edges. It is visualized as boxes connected by lines on a canvas. Data in the form of messages is passed from one object to another. An object receives a message in its inlet, processes the data, and sends a new message out of its outlet. The way that data flows through the graph determines the patch's behavior. Max/MSP is based on a reactive dataflow programming model. The graph automatically re-executes when events such as the modification of a node or edge occur, resulting in a different outcome. This creates the live programming experience.

A Java external is written to extend Max/MSP with robot control functionalities. It consists of a package of five classes (see Figure 1); three of them – ScriptBuilder, Listener and Sender can be directly assessed in the Max/MSP environment through the mxj object; the remaining two classes provide robot communication and command generation functionalities. As a proof of concept, a robotic folding process is programmed and controlled in Max/MSP. A Universal Robot UR5 robotic arm is used. Equipped with a pneumatically actuated gripper, it positions 1mmm thick aluminium strips in an automated clamp and folds them into different configurations. This process was originally explored by students in an architecture studio at the Future Cities Laboratory (Lim, Gramazio & Kohler, 2013), and is further developed here.

Figure 1: UML diagram of LiveFold Java package/external

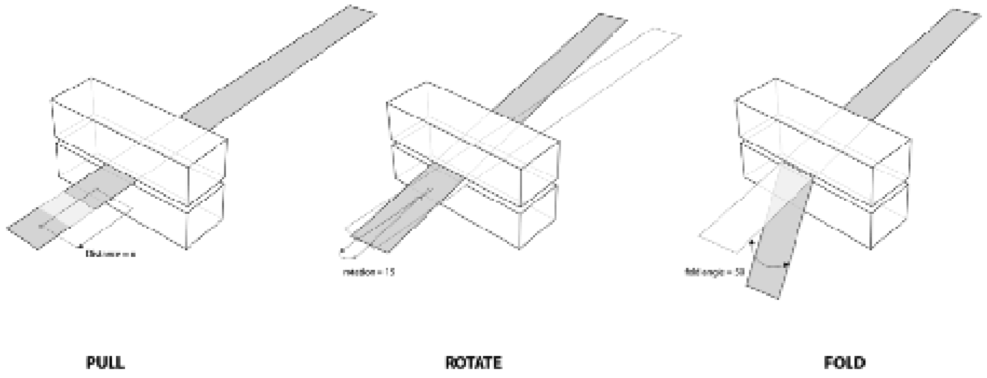


Creating the patch

The initial version of the patch consisted of message, mxj and button objects. Three main messages (on, off and move) are the fundamental building blocks of the program. On and off messages open and close the robot's gripper and the actuated clamp, while move messages control the robot's motions. The three mxj objects are scriptbuilder, sender and listener. The scriptbuilder object interprets received messages and builds a list of robot commands. This list is passed on to the sender object, which automatically opens a socket to the robot and forwards it the commands. Meanwhile, the listener object tracks the real-time state of the robot. The buttons are mainly used to trigger multiple messages at the same time.

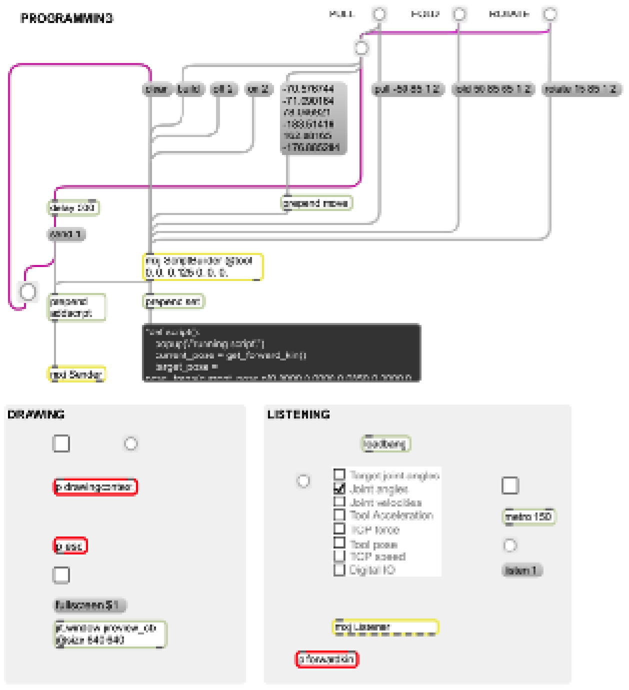
Robot actions were prototyped by assembling together different move and on/off messages. These messages are connected to the scriptbuilder object, which, in turn, is linked to the sender object. Any changes made to this graph, either to its nodes (adding, modifying or deleting objects) or edges (creating, deleting or re-routing patch-cords), triggers a live robotic response. Three types of actions were eventually prototyped: pull, rotate and fold (figure 2). Each action is a collection of move and on/off messages sequenced in a different way. At this point, a decision was made to encapsulate each action as a new kind of message. To do this, pull, rotate and fold functions were added to the Scriptbuilder class. Subsequently, these new messages became the fundamental building blocks of the patch.

Figure 2: Three key robotic actions



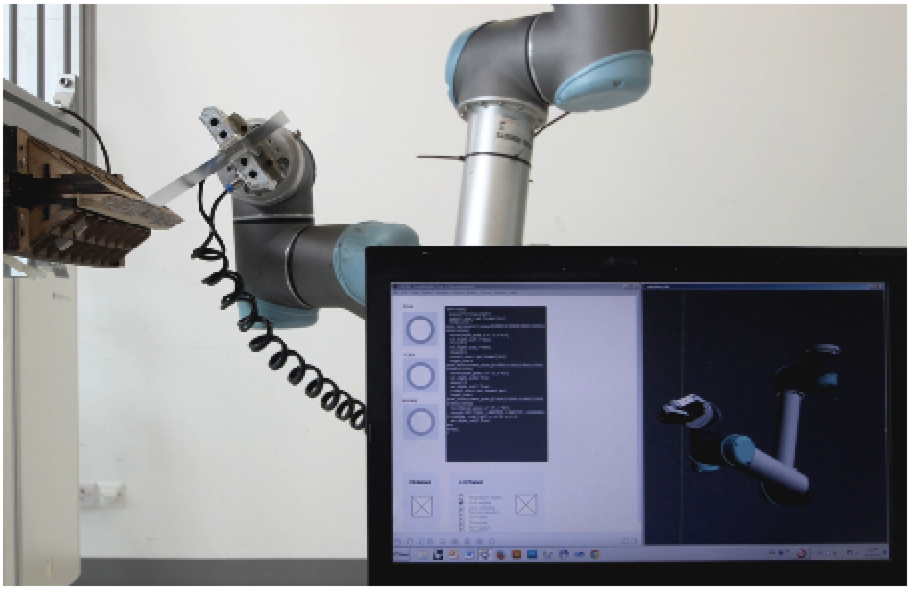
The final version of the patch is composed of three functional groups: programming, listening and drawing (figure 3). The programming group is similar to initial patches, except that pull, fold and rotate messages are primarily used in place of move and on/off messages. The listening group is responsible for updating a virtual kinematic model of the robot based on its real-time state. The drawing group creates a window for visualizing this virtual robot. The listening and drawing groups are not essential for controlling the robot, but were created to give the programmer additional feedback.

Figure 3: Final version of patch (edit mode)



The patch may also be run in presentation mode as a simplified interface geared for real-time control (figure 4). Important user interface objects remain on the canvas and are enlarged, while the rest of the graph from the default edit mode is hidden from view. Three main buttons are exposed that trigger pull, rotate and fold robotic actions. Based on the permutation of button-presses, different folded forms are eventually produced.

Figure 4: Patch in presentation mode (foreground) controlling the robotic folding process (background).



Benefits and drawbacks

Material behaviour plays a critical role in the folding process. To get a desired fold angle, the robot has to over-fold the aluminium strip by a certain amount, and at a specific speed and acceleration. The correct values for these parameters were unknown beforehand. With the live Max/MSP patch, different values could be tested and the results immediately assessed. Consequently, the folding action was quickly calibrated and accurate folds were produced. Therefore one key advantage of live programming is that it facilitates users in gaining robotic control over unpredictable material processes.

Live programming enabled a more exploratory approach towards patch development to be taken. This is especially beneficial when the desired program outcome is vaguely defined. During the patch development process, message objects were connected in various permutations and empirically tested. A general folding concept steered this bricoleur-style (Turkle & Papert, 1990) approach. Through these rapid exploratory cycles, several robotic actions were prototyped which could produce unanticipated aesthetic effects. Furthermore, an executing program may be interrupted at any point and the folded strip physically evaluated. This informs subsequent decision-making. For example, messages may be sent in a different sequence. As a result, a strip's final form is not pre-determined, but can be designed in a more sketch-like manner.

However, 'live-ness' also presents drawbacks. Since the sender object is 'always on', care must be exercised when editing the patch in order to prevent inadvertent robotic motions that may damage the folded artifact. In certain situations, it is desirable for the robot to go 'offline'. For example, changing a single parameter in a message object automatically triggers a robot response. Instead, a programmer may want to edit multiple objects first before having the robot react. This occurs frequently at the early programming stage, when changes to the patch take place on a structural level. One solution would be to provide programmers mechanisms for toggling between live and offline states.

Creative and educational applications of live programming

The majority of industrial robots today are still found on the factory floor performing tasks that are repetitive and predictable. For operators of these machines, live programming offers no significant advantages over standard online or offline approaches. The question therefore arises: who are the robot users that would benefit from adopting a live programming approach?

One such group is comprised of musicians, artists and architects engaged in robotics. These users are primarily interested in creating unique robotic performances or robotic fabricated artefacts. A less prescriptive mode of human-robot interaction robot may open up new creative avenues for exploration. For example, architects could work with material systems such as sand (Gramazio, Kohler & Willman, 2014, p. 286-301) or foam (Gramazio, Kohler & Willman, 2014, p. 84-99) with unpredictable behaviour. In these cases, some form of human decision-making or correction during the physical process is desirable. The staging of performances (Bory, 2009) involving humans and industrial robots is also becoming a subject of artistic interest and inquiry. Through live programming, artists may mediate performances in response to audience feedback and thus design more interactive experiences.

Novice robot users constitute a second group. Live programming encourages a constructivist mode of learning whereby experimentation with code changes is rewarded by immediate robot responses. The cognitive gap between looking at code and understanding its effect is reduced (Burckhardt et al., 2010). This facilitates users in acquiring robotics knowledge and programming skills. While teaching the robot what to do, the novice is simultaneously learning more about the machine and how better to communicate with it. More open-ended forms of human-robot interaction may encourage new types of users to engage in robotics; and consequently prompt the migration of these machines from their traditional industrial setting to everyday workplaces.

Acknowledgements

This work was established at the Singapore-ETH Centre for Global Environment Sustainability (SEC), co-funded by the Singapore National Research Foundation (NRF) and ETH Zurich. I would like to thank the following people for their help: Professor Fabio Gramazio (ETH Zurich) for discussing with me the topic of live programming in relation to robotic fabrication; Associate Professor Steven Miller (National University of Singapore) for his lessons on Max/MSP; and former students: Martin Tessarz, Sebastian Ernst, Silvan Strohbach and Sven Rickhoff for introducing the robotic folding process; and Chiang Pun Hon for building the robot gripper and clamp that was used in the case study example.

References

Biggs, G.M., & MacDonald, B. (2003). A survey of robot programming systems. In J.Roberts & G. Wyed (Eds), *Proceedings of the 2003 Australasian Conference on Robotics and Automation* (CSIRO). Brisbane. Retrieved from http://www.araa.asn.au/acra/acra2003/papers/27.pdf

Bökesoy, S., & Adler, P. (2011). 1city1001vibrations: Development of a interactive sound installation with robotic instrument performance. In A.R. Jensenius, A. Tveit, R.I. Godøy & D. Overholt (Eds.), *Proceedings of the International Conference on New Interfaces for Musical Expression* (pp. 52-55). Oslo: University of Oslo and Norwegian Academy of Music.

Bory, A. (2009). *Compagnie 111: Sans objet.* Retrieved from http://www.cie111.com/

en/111/performances/creations/sans-objet (25.6.2012).

Burckhardt, S., Fahndrich, M., de Halleux, P., Kato, J., McDirmid, S., Moskal, M. & Tillmann, N. (2013). It's alive! Continuous feedback in UI programming. In H.J. Boehm & C. Flanagan (Eds), *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation* (pp. 95-104). New York, NY: ACM New York.

Diaz, F. (2012). Outside itself: Interactive insallation assembled by robotic machines untouched by human hands. In S. Brell-Cokcan & J. Braumann (Eds), *ROB|ARCH Robotic fabrication in architecture, art, and design* (pp. 180-183). Vienna: Springer Wien New York.

Ge Wang. (2010). *The ChucK audio programming language: A strongly-timed and on-the-fly Environ/mentality* (Doctoral dissertation). Princeton University. Retrieved from https://ccrma.stanford.edu/~ge/thesis.html

Gramazio, F., & Kohler, M. (2008). *Digital materiality in architecture.* Baden: Lars Müller Publishers.

Gramazio, F., Kohler, M., & Willman, J. (2014). *The robotic touch: How robots change architecture.* Zurich: Park Books AG.

Hancock, C. M. (2003). *Real-time programming and the big ideas of computational literacy* (Doctoral dissertation). Massachusetts Institute of Technology. Retrieved from http://llk.media.mit.edu/papers/ch-phd.pdf

Lim, J., Gramazio, F., & Kohler, M. (2013). A software environment for designing through robotic fabrication. In R. Stouffs, P. Jansen, S. Roudavski & B. Tuncer (Eds), *Open systems: Proceedings of the 18th International Conference on Computer-Aided Architectural Design Research in Asia* (pp. 45-54). Singapore: National University of Singapore.

McCarthy, J. (2002). Rethinking the computer music language: Supercollider. *Computer Music Journal, 26*(4), 61-68.

Puckette, M. (1996). Pure data. In *Proceedings of International Computer Music Conference* (pp. 37-41). Ann Arbor, MI: MPublishing University of Michigan Library.

Puckette, M. (2002). Max at seventeen. *Computer Music Journal, 26*(4), 31-43.

Sorensen, A. & Gardner, H. (2010). Programming with time: cyber-physical programming with impromptu. In W.R. Cook, S. Clarke, M. Rinard, K.J. Sullivan & D.H. Steinberg (Eds), *Proceedings of the ACM international conference on Object oriented programming systems languages and applications* (pp. 822-834). New York: ACM New York.

Turkle, S. & Papert, S. (1992). Epistemological pluralism and the revaluation of the concrete. In I. Harel & S. Papert (Eds), *Constructionism* (pp. 161-191). Norwood, NJ: Ablex Publishing Corporation.