Improv: Live Coding for Robot Motion Design

Alexandra Q. Nilles
Department of Computer Science
University of Illinois at Urbana-Champaign
nilles2@illinois.edu

Chase Gladish
Department of Computer Science
University of Illinois at Urbana-Champaign
gladish2@illinois.edu

Mattox Beckman
Department of Computer Science
University of Illinois at Urbana-Champaign
mattox@illinois.edu

Amy LaViers
Mechanical Science and Engineering Department
University of Illinois at Urbana-Champaign
alaviers@illinois.edu

ABSTRACT

Often, people such as educators, artists, and researchers wish to quickly generate robot motion, but the tools available for programming robots can be difficult to learn, especially for people without technical training. This paper presents the *Improv* system, a programming language for high-level description of robot motion with immediate visualization of the resulting motion on a physical or simulated robot. *Improv* includes a "live coding" wrapper for ROS (“Robot Operating System", an open-source robot software framework which is widely used in academia and industry, and integrated with many commercially available robots). *Improv* also includes a domain-specific language, which is compiled to ROS messages. The language is inspired by choreographic techniques, which allows for several ways of composing and transforming movements in space and time. In this paper, we present our work on *Improv* so far, as well as the design decisions made throughout its creation.

MOCO, June 28–30, 2018, Genoa, Italy
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6504-8/18/06.
https://doi.org/10.1145/3212721.3212882

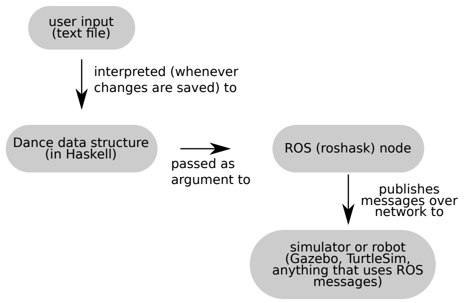

user input
(text file)
interpreted (whenever changes are saved) to
Dance data structure
(in Haskell)
passed as argument to
ROS (roshask) node
publishes messages over network to
simulator or robot
(Gazebo, TurtleSim, anything that uses ROS messages)

Figure 1: An illustration of how user input, written to a text file, is converted into a ROS node which publishes messages to a simulator or physical robot.

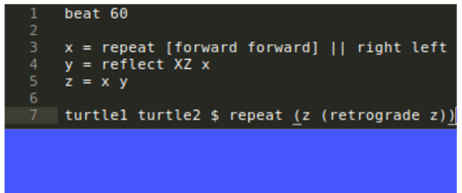
The following ROS Python client code will cause a mobile robot such as a Roomba or Turtlebot to follow a path that curves forward and left:

```
if __name__ == '__main__':
    pub = rospy.Publisher('turtle1/cmd_vel',Twist)
    rospy.init_node('publisher_node')
    loop_rate = rospy.Rate(5)
    while not rospy.is_shutdown():
        vel=Twist()
        vel.linear.x = 1.0
        vel.angular.z = 1.0
        pub.publish(vel)
        loop_rate.sleep()
```

The equivalent code in *Improv* is

```
turtle1 $ forward || left
```

where || is an operator which combines movements in parallel.



```
beat 60

x = repeat [forward forward] || right left
y = reflect XZ x
z = x y

turtle1 turtle2 $ repeat (z (retrograde z))
```

CCS CONCEPTS

• **Computer systems organization** → **External interfaces for robotics**; • **Applied computing** → *Arts and humanities*;

KEYWORDS

robotics, choreography, live coding, ROS, Haskell, roshask, HRI

ACM Reference Format:
Alexandra Q. Nilles, Chase Gladish, Mattox Beckman, and Amy LaViers. 2018. *Improv*: Live Coding for Robot Motion Design. In *MOCO: 5th International Conference on Movement and Computing, June 28–30, 2018, Genoa, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3212721.3212882

INTRODUCTION

Robotic technology is becoming more commonly integrated into settings outside of the factory - including classrooms [14] and art installations [22]. In many of these cases, users often do not have extensive programming experience, and the tasks required of the robots do not involve complicated logic. The time is ripe for *choreographic* methods of programming robots - specification languages which more closely match our mental models of motion.

Currently, many commercially available robots are programmed through interfaces created for each specific robot by the manufacturer (which may be graphical, text-based, or physically interactive) or through ROS (the "Robot Operating System,") [20] [19]. We target an improved user experience for ROS, because it is a free and open source toolset which is compatible with many platforms. *Improv* is essentially a wrapper around ROS. This gives us the benefits of ROS's infrastructure, but we exchange the powerful low-level control available in most ROS client libraries for the simplicity of a high-level representation of robot motion.

The ROS workflow has two obstacles that often make robot programming difficult, especially for newcomers to the field. First, the programming languages and interfaces are often at a low level of abstraction, requiring users to painstakingly translate their mental model of the intended movement. Second, the process of writing code, compiling and executing the instructions on the robot platform can be time-intensive and intimidating. For example, a beginner tutorial for ROS will have the user open at least three terminal windows, a text editor, and a robot simulator, for a total of five windows. It is often not possible to see all the relevant windows at one time on a regular computer monitor, making it difficult for the user to have a coherent mental model of information flow in the system.

The tool introduced in this paper, *Improv*, addresses both of these problems. To address the first (mismatch between the problem and program domains), we introduce a small domain-specific programming language (DSL) with motion primitives and operators which allow movements to be combined and transformed in space and time. The transformations are inspired by choregraphic techniques, such as those in [13] [7] [2] and [12]. To address the second obstacle to robot programming (the difficult process of evaluating code on a robot), we introduce a "live coding" infrastructure so the user can observe the effects of their instructions on the simulated or physical robot nearly immediately.

By addressing these two shortcomings of current robot programming tools, we hope to make robotics more accessible and usable for a broader range of people, not just expert roboticists. Possible users of this tool include artists, educators, newcomers to robotics, and anyone who wishes to quickly prototype robot motion patterns. *Improv* is open-source and available at https://github.com/alexandroid000/improv. Please let us know if you try it out!

Related Work. Especially when used with the two-dimensional Turtlesim, *Improv* is reminiscent of *Logo* [18], an educational language that is often used in conjunction with a simulation of a two-dimensional turtle. Our programming language is less expressive and powerful than *Logo*, but is integrated with ROS and thus able to be used with three-dimensional simulators and actual robots. Scratch, an educational, visual programming language has been integrated with ROS[1], however Scratch does not model movement directly. We are aware of two other tools for live programming in ROS, one which uses the Python shell [1], and one which uses the Live Robot Programming language and the Pharo ROS client [4] [8]. However, the aims of these projects and *Improv* are different - their respective languages were not designed around modelling movement, and instead focus on low-level sensor and actuator commands and logical control flow. Both of these related projects are better suited for applications which involve reactivity and sensing of the environment, while *Improv* is better suited to applications where the user wishes to quickly generate certain movements and creatively explore movement patterns. *Improv* is heavily influenced by *Dance*, a domain-specific language built in Haskell [11]. The project includes a DSL inspired by Labanotation, but predated ROS. Another relevant project is *roshask* [6], a Haskell client library for ROS, which this project uses as an interface between our domain-specific language and ROS.

[1]http://wiki.ros.org/scratch

PROTOTYPING MOVEMENT DESIGN IN EMBODIED IMPROVISATION

Improv is a tool for *prototyping* robot motion. Put another way, it is a tool for improvising movement on robot platforms. The authors have taken inspiration from their experiences with embodied improvisation, which is often used by choreographers and dancers as a way of understanding and creating human movement. Experts such as William Forsythe have analyzed strategies for improvisation for choreography and performance [9]. Improvisation helps the movement designer understand and explore the plethora of movement options that are available at any given time. This is especially useful in robotics applications as the field starts to explore stylized movement and the incorporation

Algorave and Live Coding

This work is also heavily influenced by live coding interfaces and programming languages for generating music and visuals, which are often associated with the Algorave performance movement [5]. In particular, the programming language TidalCycles [16] has had a strong influence on this work, both syntactically and in how relative timing of events is managed. *Al Jazari* is a live coding installation which uses a simple graphical language to allow people to control robots (in simulation) [15]. The language includes conditionals based on external state and communication between the bots. The program state of the robot is also visualized. There are a variety of other projects centered of robotic technology into homes and artistic performances. For example, one may explore different ways of picking up a cup (as if one is underwater, or as if one is being electrocuted, or in the style suggested by different pieces of music) in order to understand how many different ways there are for a robot to perform one task, and the percieved effects of all these different motion strategies.

However, the time taken to set up environments and write, compile and execute code often negates the benefits of improvisational practice when done on a robotic platform instead of a human body. This is doubly true when working with robotic hardware, and these barriers especially affect those users who do not have a strong background in programming. This places some design constraints on the *Improv* system - namely, the system must have

- a minimal “representational distance" between the user’s mental model of the movement they want to try and the description in code, so there is minimal frustration and time wasted in translation,
- a near-imperceptible delay between writing instructions to the robot and seeing the effect of those instructions, and
- a singular environment where the user interacts with the program (to avoid the user’s attentional flow being broken by needing to switch between different interaction modalities).

Designing *Improv* with these principles in mind has helped us minimize the mental load of using the tool, and enable faster prototyping and a more improvisational workflow.

IMPROV ARCHITECTURE AND FEATURES

To specifically address these design criteria, we have included the following features in *Improv*:

- *small representational distance between movement and code:* a domain-specific language, inspired by choreographic techniques such as spatial symmetries, relative timing changes, and body-centric coordinates. Many domain-specific languages for robot programming exist; for example, a review in 2016 identified 137 relevant publications [17]. However, most model robotic systems from a control-flow perspective, and do not model motion directly. The desired language will depend heavily on the programmer’s task, but undoubtably there are many applications where users would like to describe the robot’s instructions directly in terms of movement. It then makes sense to directly use the expertise and terminology developed by choreographers and other movement experts.
- *rapid movement prototyping:* changes to the user’s file are interpreted by a Haskell program that builds a ROS node for publishing messages to a simulator or physical robot. This process is nearly real time, allowing for a seamless user experience.
- *workspace with few attentional switches:* a live coding interface with only two windows, one for editing the text file and one for observing effects on a simulated robot.

The authors were inspired by several of the principles outlined in the ‘cognitive dimensions of notations’ [10]. There are eleven ‘cognitive dimensions,’ or design principles, that the authors describe but several are especially relevant to this work, such as

– *Closeness of mapping*: "the closer the programming world is to the problem world, the easier the problem-solving ought to be. Ideally, the problem entities in the user’s task

prim	→	rest
	|	forward
	|	left
	|	halfleft
	|	right
	|	halfright
movement	→	prim
	|	movement movement
	|	[movement]
	|	(movement)
	|	movement || movement
	|	transformer movement
transformer	→	reverse
	|	retrograde
	|	repeat n
	|	reflect ax
exp	→	rs $ movement
	|	var = movement

Table 1: The grammar of *Improv* programs. exp represents top-level expressions, which execute movements on robot(s), or store movements in

All of these features are intended to give the user a sense of *flow*: a mental state of complete absorption in the activity. The fewer distractions and focus changes in the activity, whether it is improvisational dance or coding, the higher the chance of the participant becoming completely engaged and accessing all the creative options available.

As examples of possible user interfaces, Figure 2 shows an example of the system in use. Note that the choice of editor and the choice of simulator are decoupled, and *Improv* is absolutely editor-agnostic, relying only on an operating-system level script to execute changes. *Improv* is somewhat simulator-agnostic: currently it is only possible to control robots which use a `Twist` ROS messages for control, which set desired linear and rotational velocities.

DOMAIN SPECIFIC LANGUAGE (DSL) FEATURES

The base type of the *Improv* language is a movement. Movements are discretized and can be combined with each other in various ways, forming new movements. The precise way in which instructions are interpreted on a robot platform is defined by the language's translation to Haskell and the resulting messages sent over the ROS network.

Figure 1 shows the grammar of the *Improv* language. The language supports primitives such as `forward` and `right` corresponding to commands to the robot causing it to perform the expected motion. These primitives are composed in series, sequence and parallel, and transformed in time and space. Movements are organized in units of time called "beats." The base timing of beats (units per minute) can be specified by the user, and is only limited by the maximum publishing frequency of ROS and the physical constraints of the robot platform.

Users can specify a series of commands such as *move forward for one beat, turn right for one beat, move forward for one beat* with the command `forward right forward`. Movements separated by whitespace on the same line will occur in different "beats." The user can also use brackets to compress a sequence of movements into one beat, such as `[forward right forward]`, which will cause these three movements to happen in one beat. This syntax and behavior is directly inspired by *TidalCycles*, which has a similar mechanism for grouping sounds. In our implementation, bracketing n movements causes each movement to be performed n times faster, but for $1/n$ times as long, so the movement has the same spatial extent but is performed faster.

Movements can also be performed in parallel, such as `forward || right`, which will cause the robot to curve to the right as it moves forward.

We have also implemented several transformations which map a function over a movement: `repeat`, `reflect`, `reverse`, and `retrograde`. `reflect` takes a specific (body-centric) spatial axis as an argument, and returns a transformed movement. `reverse (forward right left)` is equivalent to `left right forward`. `retrograde (forward right left)` is equivalent to `right left backward`.

INTERFACING WITH ROS

Roshask is a client library for ROS, written in Haskell. It treats streams of values (such as those published and subscribed to by ROS nodes) as first class values, which allows for them to be combined and transformed in more natural ways than imperative ROS client libraries. For example, when we put `Dances` in parallel, we wish to combine two lists of motion commands with some function for parallel execution - whether this is averaging commands which affect the same body part, or additively combining them, or whatever interpretation the designer wishes for a specific robot platform. In Haskell, this is accomplished easily with the `zipWith` function, which takes two lists and a function for combining values in those lists. *Roshask* extends this expressivity to the combination and transformation of ROS message streams, making it a useful tool for implementing the *Improv* DSL.
The primitives in *Improv*, such as `forward` or `right`, are mapped to streams of ROS messages. In our implementation so far, we have mapped to the `Twist` ROS message, which specifies the robot's linear and angular velocity as two three-dimensional vectors. Velocity controllers, which often are included with commercial robots, are required to create the low-level motor controls for reaching and maintaining the desired velocities.

CONCLUSIONS AND FUTURE WORK

This paper has presented a working implementation of a domain-specific language which is interpreted as commands to a real or simulated robot. Due to fast and automated interpretation of user programs, this system allows for a very tight feedback loop while programming robots. We hope that this feature, along with a programming language which models movement directly, will decrease the cognitive load associated with robot programming. We aim to make the creative development of robot motion patterns faster, easier, and more accessible to a broader swath of potential users.

To this end, future work must include systematic studies of people's qualitative assessment of the usability of the system, as well as quantitative measures on how quickly people iterate on programs in the *Improv* language and how much the robot moves as a result. As far as we know, no similar usability studies have been performed on the more mainstream C++ and Python ROS clients. We plan to include a range of participants in our study, including people with limited programming experience and no ROS experience, as well as people familiar with ROS. From our own explorations of the tool, we have found that the experience is quite engaging, especially when using a three-dimensional physical simulator. We have included a video as supplemental information of *Improv* being used with Gazebo. We are very interested in measuring the effects of different editing and simulating environments on the user experience.

One major limitation of *Improv* is that it includes no features for controlling the robots based on sensor observations or interactions with the environment. An interesting future extension of this work would be to interface the *Improv* DSL with ROS subscribers and include the ability to react to sensor readings and environment state. Another limitation of *Improv* is the complexity of extending the DSL and ROS interface to new robot platforms. This process requires defining the conversion from *Improv* programs to ROS messages in Haskell, and may be especially tricky for robots with many body parts and degrees of freedom. Future work will involve extending the language and interface to more complicated robots and refactoring the code base as necessary to make this extension process more accessible.

Finally, we would like to emphasize that the design decisions for how *Improv* programs are realized on robot platforms are relatively arbitrary and a single robot could have a multitude of different implementations. As Thecla Schiphorst has written, "it is not technological constraints that hold us back from using technology in new ways; technology changes at a tremendous rate. Our willingness to explore beyond the constraints of our imagination has the greatest effect" [21]. We hope that the implementation described here opens up new avenues of imagination for how robot programming can become better, and more integrated into different forms of human expression.

ACKNOWLEDGEMENTS

This work is partially funded by *[redacted]* and DARPA grant #D16AP00001.

REFERENCES

[1] Sorin Adam and Ulrik Pagh Schultz. 2014. Towards Interactive, Incremental Programming of ROS Nodes. *arXiv* (2014). http://arxiv.org/abs/1412.4714

[2] Sarah Fdili Alaoui, Kristin Carlson, and Thecla Schiphorst. 2014. Choreography as mediated through compositional tools for movement: Constructing a historical perspective. In *Proc. of the 2014 International Workshop on Movement and Computing*. ACM, 1.

[3] Alan Blackwell, Alex McLean, James Noble, and Julian Rohrhuber. 2014. Collaboration and learning through live coding (Dagstuhl Seminar 13382). *Dagstuhl Reports* 3, 9 (2014), 130–168. https://doi.org/10.4230/DagRep.3.9.130

[4] Miguel Campusano and Johan Fabry. 2017. Live robot programming: The language, its implementation, and robot API independence. *Science of Computer Programming* 133 (2017), 1–19.

[5] Nick Collins and Alex McLean. 2014. Algorave: Live performance of algorithmic electronic dance music. In *Proc. of the International Conference on New Interfaces for Musical Expression*. 355–358.

[6] Anthony Cowley and Camillo J Taylor. 2011. Stream-oriented robotics programming: The design of roshask. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 1048–1054.

[7] Shannon Cuykendall, Thecla Schiphorst, and Jim Bizzocchi. 2014. Designing interaction categories for kinesthetic empathy: A case study of synchronous objects. In *Proc. of the 2014 International Workshop on Movement and Computing*. ACM, 13.

[8] Pablo Estefó, Miguel Campusano, Luc Fabresse, Johan Fabry, Jannik Laval, and Noury Bouraqad. 2014. Towards live programming in ROS with PhaROS and LRP. *arXiv preprint arXiv:1412.4629* (2014).

[9] William Forsythe. 2004. *Improvisation technologies: a tool for the analytical dance eye*. Hatje Cantz.

[10] Thomas R. G. Green and Marian Petre. 1996. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing* 7, 2 (1996), 131–174.

[11] Liwen Huang and Paul Hudak. 2003. *Dance: A Declarative Language for the Control of Humanoid Robots*. Technical Report YALEU/DCS/RR-1253. Yale University.

[12] Doris Humphrey. 1959. *The art of making dances*. Grove Press.

[13] Amy LaViers, Catie Cuan, Madison Heimerdinger, Umer Huzaifa, Catherine Maguire, Reika McNish, Alexandra Nilles, Ishaan Pakrasi, Karen Bradley, Kim Brooks Mata, et al. 2017. Choreographic and Somatic Approaches for the Development of Expressive Robotic Systems. *arXiv preprint arXiv:1712.08195* (2017).

[14] Maja J Mataric. 2004. Robotics education for all ages. In *Proc. AAAI Spring Symposium on Accessible, Hands-on AI and Robotics Education*.

[15] Alex McLean, Dave Griffiths, Nick Collins, and Geraint A Wiggins. 2010. Visualisation of live code.. In *EVA*.

[16] Alex McLean and Geraint Wiggins. 2010. Tidal–pattern language for the live coding of music. In *Proc. of the 7th sound and music computing conference*.

[17] Arne Nordmann, Nico Hochgeschwender, Dennis Leroy Wigand, and Sebastian Wrede. 2016. A Survey on Domain-Specific Modeling and Languages in Robotics. *Journal of Software Engineering in Robotics (JOSER)* 7, 1 (2016), 75–99.

[18] Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.

[19] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, Japan, 5.

[20] Gregory F Rossano, Carlos Martinez, Mikael Hedelind, Steve Murphy, and Thomas A Fuhlbrigge. 2013. Easy robot programming concepts: An industrial perspective. In *IEEE Conf. on Automation Science and Engineering*.

[21] Thecla Schiphorst. 1986. *A Case Study of Merce Cunningham's Use of the Lifeforms Computer Choreographic System in the Making of Trackers*. Master's thesis. Simon Fraser University.

[22] Huang Yi and Joshua Roman. 2017. Huang Yi & KUKA: A human-robot dance duet. (April 2017).