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

ACM ISBN 978-1-4503-6504-8/18/06.
https://doi.org/10.1145/3212721.3212882

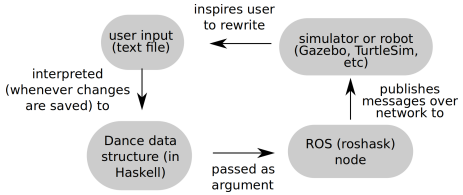

Figure 1: An illustration of how user input, written to a text file, is converted into a ROS node which publishes messages to a simulator or physical robot.

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.

CCS CONCEPTS

• **Computer systems organization → External interfaces for robotics**;

KEYWORDS

robotics, choreography, live coding, ROS, Haskell, roshask, human-robot interaction

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, 6 pages. https://doi.org/10.1145/3212721.3212882

INTRODUCTION

Robotic technology is becoming more commonly integrated into settings outside of the factory - including classrooms [15] and art installations [23]. In many of these cases, users often do not have extensive programming experience, and the tasks required of the robots require only specific motion patters and perhaps simple reactivity. The time is ripe for *choreographic* methods of programming robots, which 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,") [21] [20]. 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 for newcomers to robotics programming: 1) programs are written at a low level of abstraction, requiring users to painstakingly translate their mental model of the intended movement, and 2) the process of writing code, compiling and executing the instructions on the robot platform can be 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, 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 sticking points. With *Improv* we hope to help make robotics more accessible to a broader range of people. 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!

Live coding and algorave

This work is 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 [17] has had a strong influence on the structure of the *Improv* programming language, 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) [16]. 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 around live coding interfaces for controlling cyberphysical systems and visual simulators [3].

One important design decision for developers of interactive text-based programming tools is whether to tie their tool to a specific text editor. We decided to allow users flexibility to choose their editor of preference. Instead of creating an interface for each desired editor, we use a shell script which monitors the file that the user is editing for changes. Every time the user saves changes to the file, the program detects a change, interprets the user's new program, and resets the simulator and ROS node. This design choice circumvents the need to interface with specific editors. While we have not done a formal timing analysis, the delay is a small fraction of a second and not noticeably longer than the time it takes to look from the text editor to the simulator.

Related Work. Especially when used with the two-dimensional Turtlesim, *Improv* is reminiscent of *Logo* [19], 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 [7], which is the most closely related work to *Improv*. Our interface is textual, while Scratch is visual, and the *Improv* programming language is more focused on modelling of choreographic concepts (such as relative timings of movements and body symmetries) while Scratch is focused on game development.

Among many programming languages for robotics [18], 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] [9]. However, these languages focus on low-level sensor and actuator commands and logical control flow, rather than modeling movement. These tools are better suited for applications which involve sensing 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 inspired by Labanotation and built in Haskell [12]. 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.

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, and the creative movement design it enables. Movement experts have analyzed strategies for improvisation for choreography and performance [10]. 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 of robotic technology into homes and artistic performances.

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. 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

The authors were influenced by several of the principles outlined in the 'cognitive dimensions of notations' [11]. 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*: "Ideally, the problem entities in the user's task domain could be mapped directly onto task-specific program entities, and operations on those problem entities would likewise be mapped directly onto program operations" [11]
- *Diffuseness*: How many symbols or graphic entities are required to express a meaning?
- *Error-proneness*: Does the design of the notation induce 'careless mistakes'?
- *Hard mental operations*: Are there places where the user needs to resort to fingers or pencilled annotation to keep track of what's happening?
- *Progressive evaluation*: Can a partially-complete program be executed to obtain feedback on 'How am I doing'?

- 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).

IMPROV FEATURES

To specifically address these design criteria, we have included the following features in *Improv*. 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 available creative options.

- *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. The systems and terminology developed by choreographers and other movement experts are invaluable in this attempt, such as in [14] [8] [2] and [13].
- *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 at most, one for editing the text file and one for observing effects on a simulated robot.

Domain Specific Language (DSL) Features. The base type of the *Improv* language is a movement. Movements can be combined with each other in various ways, forming new movements. Table 1 shows the grammar of the *Improv* language. The language supports primitive robot movements such as `forward` and `right`. Movements are organized in units of time called "beats." The base timing of beats (units per minute) can be specified by the user. Movements can be composed and stored in variables. The following table shows some example programs in *Improv*.

Natural Language	Code	Comments
move forward for one beat, turn right for one beat, move forward for one beat	`forward right forward`	performed in three beats
move forward, right, and forward, all in one beat	`[forward right forward]`	performed in one beat - same spatial extent, but faster
curve right and forward	`forward || right`	
do movement `x`*, reflected across saggital plane*	`reflect YZ x`	
reverse the movement "forward right left"	`reverse (forward right left)`	same as `left right forward` - reverses the order of the primitives

```
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 variables. Movements are converted into ROS message streams and can be composed and grouped in multiple ways.

CONCLUSIONS AND FUTURE WORK

Future work must include systematic studies of the usability of the system, as compared to other tools such as Scratch and Python or C++ ROS clients. 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.

One major limitation of *Improv* is that does not incorporate sensor feedback. 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 incorporating new robot platforms. Currently it is only possible to control simple, Roomba-like robots by setting linear and rotational velocities. Extending *Improv* to more articulated robots 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" [22]. We hope that the implementation described here opens up new avenues of imagination for how robot programming can become more easily 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.

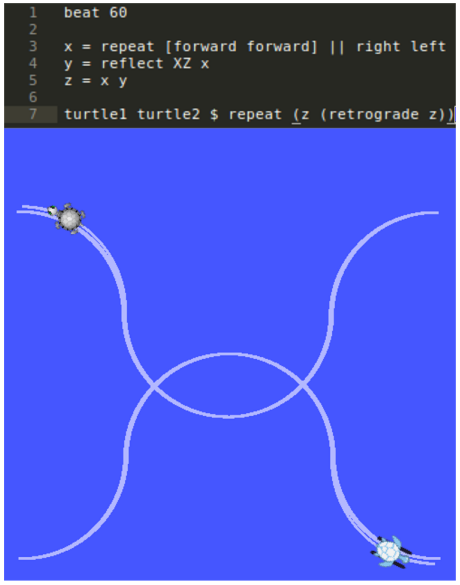

Figure 2: An example of a text-editor and simulation environment configuration available to users of *Improv*. Any text editor can be used, while simulators or robots must be compatible with the ROS message types implemented with the system.

[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] Christopher Crick, Graylin Jay, Sarah Osentoski, Benjamin Pitzer, and Odest Chadwicke Jenkins. 2017. Rosbridge: Ros for non-ros users. In *Robotics Research*. Springer, 493–504.
[8] 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.
[9] 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).
[10] William Forsythe. 2004. *Improvisation technologies: a tool for the analytical dance eye*. Hatje Cantz.
[11] 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.
[12] Liwen Huang and Paul Hudak. 2003. *Dance: A Declarative Language for the Control of Humanoid Robots*. Technical Report YALEU/DCS/RR-1253. Yale University.
[13] Doris Humphrey. 1959. *The art of making dances*. Grove Press.
[14] 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).
[15] Maja J Mataric. 2004. Robotics education for all ages. In *Proc. AAAI Spring Symposium on Accessible, Hands-on AI and Robotics Education*.
[16] Alex McLean, Dave Griffiths, Nick Collins, and Geraint A Wiggins. 2010. Visualisation of live code.. In *EVA*.
[17] 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*.
[18] 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.
[19] Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.
[20] 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.
[21] 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*.
[22] 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.
[23] Huang Yi and Joshua Roman. 2017. Huang Yi & KUKA: A human-robot dance duet. (April 2017).