Visual Robot Programming for Generalizable Mobile Manipulation Tasks

Sonya Alexandrova, Zachary Tatlock and Maya Cakmak
University of Washington. Computer Science & Engineering
185 Stevens Way, Seattle, Washington, USA
{sonyaa,ztatlock,mcakmak}@cs.washington.edu

ABSTRACT

General-purpose robots present the opportunity to be programmed for a specific purpose *after* deployment. This requires tools for end-users to quickly and intuitively program robots to perform useful tasks in new environments. In this paper, we present a flow-based visual programming language (VPL) for mobile manipulation tasks, demonstrate the generalizability of tasks programmed in this VPL, and present a preliminary user study of a development tool for this VPL.

Categories and Subject Descriptors

H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors, software psychology*

General Terms

Design, Human Factors

1. INTRODUCTION

It is unfeasible to fully program a robot before it is deployed in its context of operation. Instead, we aim to develop tools that enable intuitive, robust, and fast programming of robots *after* deployment. To that end, we apply techniques from the field of End-User Programming (EUP) to robot programming. The goal of EUP is to enable users to create programs that meet their particular needs without any traditional software development skills [4, 3]. In this paper, we present an exploration of a powerful EUP technique called *visual programming languages* (VPL) [1].

VPLs allow users to create or modify programs by graphically manipulating a program. Several VPLs have been developed for simple educational robots [5, 2]; however, the level of abstraction for these VPLs is too low for programming large-scale mobile manipulation tasks. In this paper we present an initial exploration of a new VPL called RoboFlow that is developed for programming real-world mobile manipulation tasks on a high degree-of-freedom robot (PR2).

2. ROBOFLOW

RobotFlow is a flow-based VPL, similar to *flow diagrams*. It uses a box-line representation. Boxes are procedures with one input and multiple outputs. Lines represent flow of data from the output of a box to the input of another box. It does not involve control flow constructs such as *while*, instead allowing loops through cycles in the flow graph. Each program has a start terminal, procedures, at least one success end-terminal and possibly failure end-terminals.

HRI'15 Extended Abstracts, March 2–5, 2015, Portland, OR, USA.
ACM 978-1-4503-3318-4/15/03. http://dx.doi.org/10.1145/2701973.2702052.

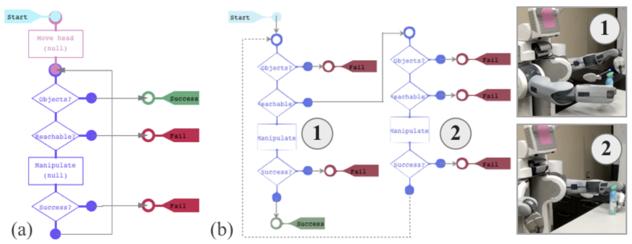

Figure 1: Sample programs in RoboFlow that demonstrate (a) looping and (b) branching.

Based on the capabilities required for mobile manipulation we include three types of procedures that control different actuators on the robot: (i) manipulation, (ii) navigation, and (iii) active perception (*i.e.* head movements). The core is the *operation*, which is a low-level subroutine that interacts with the actuators. The operation may have *pre-conditions* to be checked before execution and *post-conditions* to be checked after execution. A program can be abstracted as a procedure by considering its start as input and its two types of end-terminals as its two outputs.

Programming in RoboFlow. To program a task, users first demonstrate one execution of the program. This is used to automatically create a *default* linear program that has all demonstrated procedures in a sequence, fails if any condition is not met and succeeds only when the last procedure completes successfully. Users then edit the program structure through a graphical interface. They can change the program to repeat or skip certain parts of the default program in some conditions. Users can also change object similarity thresholds in order to allow demonstrated manipulation procedures to be applied on different objects. Note that programs could also be created from scratch through RoboFlow, but that requires a mechanism for instantiating each procedure manually added to the program.

Example Programs. We present two example RoboFlow programs. Fig. 1(a) shows a basic looping program, such as picking up objects and putting them in a box or stacking cups. This program continues as long as suitable objects are present. Fig. 1(b) demonstrates branching: if an object

Table 1: Task metrics in the user study: completion time (seconds) and the number of errors made in the task.

Participant	Comprehension		Debugging		Creation	
	time	*#err*	*time*	*#err*	*time*	*#err*
P1 (Robo)	110	0	130	0	235	1
P2 (Robo)	255	0	245	0	245	1
P3 (Robo)	115	0	170	0	505	0
P4 (PL)	95	0	135	0	240	0
P5 (PL)	160	0	235	0	190	0
P6 (PL)	130	1	205	0	205	0
Average	*144*	*0.16*	*187*	*0.00*	*270*	*0.33*
St.dev.	*59*	*0.41*	*49*	*0.00*	*117*	*0.52*

is reachable by the right arm, pick it up, otherwise push it right with the left arm until it becomes reachable. RoboFlow features such as conditions, branching, nesting and looping make it possible to generalize across object numbers, types and conditions, while also handling dynamic changes in the environment and errors.

3. USER STUDY

We present a pilot user study that explores the *comprehension*, *debugging*, and *creation* of programs in RoboFlow.

Procedure. Participants are introduced to the RoboFlow GUI through an example. The experimenter introduces the three types of procedures explaining their operation and demonstrates adding them to the program. Participants do not instantiate procedures, but choose among existing instantiations (*e.g.* "pick up small object with left arm" or "look at the table"). The experimenter then creates a sample program demonstrating how links are created and edited.

In the *comprehension* task, participants are shown a program (Fig. 1(a)) and are asked to describe how the program would behave. Their description is timed and recorded. In the *debugging* task participants are given a *default* program that is automatically created from a single demonstration (Fig. 1(c)). The desired program behavior is described and participants are told to modify the program so it would behave as intended. The *default* program is missing an edge that results in repeating the pushing procedure until the object is reachable by the right arm. Participants need to correct that edge. In the *creation* task participants create a program from scratch for a described behavior: robot searches for a particular object around the lab and picks it up when it is found. At the end, we conduct a semi-structured interview with the participants to get their feedback on RobotFlow.

3.1 Findings

Our user study included 6 participants (5 male and 1 female, ages 24-28). Three were roboticists and three were programming languages experts. Although RoboFlow is intended for a diverse group of users, initial evaluation was restricted to these extreme user types, as their insights can be valuable in improving RoboFlow before a large scale user study. Table 1 summarizes the metrics from the user study.

Overall performance. After only a four minute tutorial on RoboFlow, all users completed each task in a few minutes, making very few mistakes. One of the programming languages experts misunderstood the relationship between looking (changing head pan and tilt) and object detection during the Comprehension test. Two of the robotics experts made small mistakes during the Creation test where their program instructed the PR2 to continue searching for an object even after the specified task required failure. Note that out of all 18 tests, only one instance of misunderstanding arose. We conclude that many programmers will find RoboFlow to be highly intuitive. Furthermore, with the exception of one test, all tasks were completed in roughly four minutes or less. The one exception (P3, Creation) was an outlier as that user decided to recreate their program when it was nearly complete to "make the line layout more clear".

Difference between tasks. As expected, the Comprehension, Debugging, and Creation tasks were ordered by amount of time required to complete the task. However, this comparison is not conclusive, as it does not take into account the difference between the programs in each task. The creation task also involves more mechanical steps (dragging tokens into the program and connecting them) whereas debugging involves just a few edits on an existing program. Hence, improvements on the editor that streamline mechanical tasks could reduce the gap between the two tasks.

Difference between two user types. While the task metrics show similar quantitative performance for the two groups, their approach to solving the various tasks was distinct. The robotics experts tended to describe their reasoning in real world terms as they worked the tasks, using phrases like "When the robot moves between tables, if it encounters an obstacle, then the whole task should fail". In contrast, the programming language experts often focused on invariants arising from the path constraints required to reach a certain procedure in the program graph, *e.g.* "At this point I know the object cannot be present since all paths to this node establish and maintain that invariant."

4. CONCLUSION AND FUTURE WORK

Our user study yielded promising results, showing that RoboFlow is intuitive. Next, we plan to evaluate RoboFlow as a complete system (including procedure instantiation) with a larger population of non-programmers.

5. REFERENCES

[1] M.M. Burnett. Visual programming. *Wiley Encyclopedia of Electrical and Electronics Engr.*, 1999.

[2] S.H. Kim and J.W. Jeon. Programming lego mindstorms nxt with visual programming. In *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*, 2468–2472. IEEE, 2007.

[3] A.J. Ko, B.A. Myers, and H. H. Aung. Six learning barriers in end-user programming systems. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 199–206. IEEE, 2004.

[4] H. Lieberman, F. Paterno, M. Klann, and V. Wulf. *End-user development: An emerging paradigm.* Springer, 2006.

[5] F. Riedo, M. Chevalier, S. Magnenat, and F. Mondada. Thymio ii, a robot that grows wiser with children. In *Advanced Robotics and its Social Impacts (ARSO), 2013 IEEE Workshop on*, pages 187–193. IEEE, 2013.