Programming Robot Manipulators with Tangible Blocks

Yasaman S. Sefidgar
Paul G. Allen School of Computer Science and Engineering
University of Washington
einsian@gmail.com

Maya Cakmak
Paul G. Allen School of Computer Science and Engineering
University of Washington
mcakmak@cs.washington.edu

Abstract

While the cost of making industrial robots declines, their deployment remains expensive. Widespread use of robots, particularly in smaller industries, is more easily realized if robot programming is accessible to non-programmers. Our research explores techniques to lower the barrier to robot programming. One such technique is *situated tangible programming* to program a robot by placing specially designed tangible *blocks* in its workspace. These blocks are used for annotating objects, locations, or regions, and specifying actions and their ordering. The robot compiles a program by detecting blocks and objects in the environment and grouping them into instructions by solving constraints. We designed a tangible language and the associated blocks and evaluated the intuitiveness and learnability of the approach. Our user studies provide evidence for the promise of situated tangible programming and identify the challenges to address. In addition to improving the block design and extending the language, we are planning to integrate tangible programming into a holistic ecosystem of a programming environment.

Keywords End-User Robot Programming, Programming by Demonstration, Tangible Programming

1 Introduction

Traditional automation is too costly and inflexible to satisfy the low volume and high mix production requirements that are commonly encountered in smaller industries. Becoming increasingly cheaper and safer, robotic manipulators hold the promise for revolutionizing industrial automation. Lower cost and programmability of robots allow industries of all sizes to benefit from automation and respond to dynamic customer needs. There is, however, a substantial barrier to realizing such promise: the languages and interfaces for programming robot manipulators are notoriously complex. Programming robots thus remains expensive, as it requires advanced knowledge and expertise and takes considerable time even for trained experts.

Programming by Demonstration (PbD) is a popular approach to make robot programming accessible to non-experts.

PLATEAU'17 Workshop on Evaluation and Usability of Programming Languages and Tools, October 23, 2017, Vancouver, CA

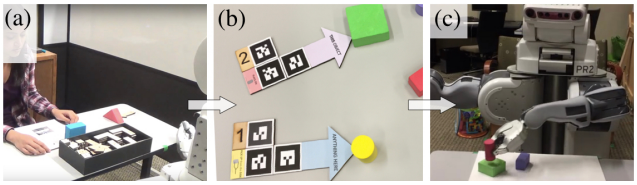

Figure 1. *(a) Situated tangible programming* involves programming a robot by combining and placing specially designed tangible blocks in the robot's workspace to select objects, locations, or regions, and to specify actions (e.g. pick or place) and their ordering. (b) Blocks and objects in the workspace are detected by the robot and compiled into a robot program. (c) The robot can perform the instructed task in new environments by executing this program. Once compiled, the blocks can be removed from the environment; program execution does not require the blocks to be present.

Despite considerable research in this area, there remains difficulties to address. Referencing objects or arbitrary parts of the environment where robots' actions take place poses a particularly difficult end-user programming challenge. Some programming systems incorporate a separate procedure to specify objects or locations that are relevant for the task and typically require the definition of coordinate frames in relation to the environment. Others have proposed "situated" approaches, such as using pointing gestures [Fang et al. 2015], verbal descriptions [She et al. 2014], or visual annotations [Fung et al. 2011], but lack the robustness needed in industrial settings.

In our research we explore a new way of programming robots that is both robust and takes advantage of being situated in the task context. It is also expressive enough for the requirements of a wide range of industrial tasks. Our approach involves placing physical, tangible *blocks* in the robot's task environment to annotate objects, locations, or regions and to instruct the robot to perform actions in relation to those annotations. Next section further details this new programming technique.

2 Situated Tangible Programming

We focus on programming pick-and-place tasks in industrial settings and consider pick and place targets common in such

settings: fixed locations or regions, as well as single or multiple specific objects. For example, a robot may obtain an object from a feeder (fixed location) and throw it in a bin (fixed region). We also consider different ways of picking (from the top of an object as opposed to its side) and placing (placing on a surface or dropping above). We assume the robot can detect the surfaces on which locations, regions, and objects of interest are found.

In our framework, a *program* is a sequence of instructions to accomplish a certain pick-and-place task when executed by the robot. The robot's actions are represented with two functions:

- `pick-up-from-top/side(location` ℓ`)`: Picks up the object at ℓ from the top/side of the object. No action is executed if no object exists at ℓ.
- `place-at/drop-above(location` ℓ`)`: Places/drops currently held object at/above location ℓ. No action is executed if the robot's gripper is empty.

Our programming language has three data structures:

- A `location` ℓ is a 3-dimensional point on a surface.
- A `descriptor` d is an object's model that allows the object to be localized in robot's workspace.
- A `region` R is a set of points in a convex hull on a surface.

The language also supports *arrays* of object descriptors and locations. A program has both *constants* and *variables* of these data structures. *Constants* are given as input to the program at programming time and hold information about known object descriptors, locations, and regions. Two variables are instantiated at programming time: an array of descriptors ($\mathbf{D} = [d_i]_{i=1}^{N}$) and an array of locations ($\mathbf{L} = [\ell_j]_{j=1}^{N}$) corresponding to objects present at execution time.

Our language additionally supports *for-loops*, *conditionals*, and the following convenience operations:

- `is-same(`d_1`,` d_2`)` compares two object descriptors and determines if they are perceptually equal.
- `is-in(`ℓ`,` R`)` checks if `location` ℓ is in `region` R.
- `find(`d`,` $\mathbf{D}$`)` returns the index i of a `descriptor` $d_i \in \mathbf{D}$ where `is-same(`d`,` d_i`)=True`. For multiple matches, one is selected randomly. The function returns `Null` if no match is found.
- `random(`R`)` returns a random location in `region` R.

This language allows a range of object manipulation programs and is extensible, without any change in semantics, to other types of robot actions that a different end-effector can perform (e.g. drilling or welding) so long as the action is defined on a location, region, or object. A simple program for picking up an object at a known location and placing it on a certain object is given in Algorithm 1 (Fig. 1-b). If this program were to be created manually, specifying the pickup location and the object descriptor would require a separate procedure or tool. The programmer cannot simply guess the coordinates of the location or the values in the descriptor.

Algorithm 1: A program for picking up from a known location and placing on a specific object (Fig. 1-b).

Input : `location` *pickup-location*
`descriptor` *green-cube*
Output : Execution of actions on the robot

```
1 [descriptor] D;
2 [location] L;
3 D, L = perceive-workspace();
4 int j=find(green-cube, D);
5 pick-up-from-side(pickup-location);
6 place-at(L[j]);
```

We designed tangible blocks that allow programming in this language and implemented a proof-of-concept perception and execution system that compiles tangible instructions to robot motions [Sefidgar et al. 2017]. Briefly, we considered *selection*, *action*, and *ordering* blocks that once put together realize pick-up or placement at various targets.

3 Evaluations

We conducted three user studies to examine the intuitiveness and learnability of the language and the associated blocks, and to identify areas for improvement. More specifically, we examined whether people can understand tangible instructions or create programs with them. We compared program comprehension and program creation performance before and after training in our first study and evaluated the same capabilities without any training in two follow-up studies [Sefidgar et al. 2017]. We found that situated tangible programming allows participants to program a robot with minimal or no instruction. We cannot draw a direct comparison with instructions given to participants for other end-user robot programming techniques; however, participants' ability to program the robot without *any* instructions is unique. Further comparative studies will highlight the advantages and limitations of this approach against others.

References

Rui Fang, Malcolm Doering, and Joyce Y Chai. 2015. Embodied collaborative referring expression generation in situated human-robot interaction. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 271–278.

Richard Fung, Sunao Hashimoto, Masahiko Inami, and Takeo Igarashi. 2011. An augmented reality system for teaching sequential tasks to a household robot. In *2011 RO-MAN*. IEEE, 282–287.

Yasaman Sefidgar, Prerna Agarwal, and Maya Cakmak. 2017. Situated Tangible Robot Programming. In *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*.

Lanbo She, Yu Cheng, Joyce Y Chai, Yunyi Jia, Shaohua Yang, and Ning Xi. 2014. Teaching robots new actions through natural language instructions. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*. IEEE, 868–873.