Extended State Automata for Intuitive Robot Programming*

Lukas Sauer[0000−0002−7808−0907] and Dominik Henrich

Universität Bayreuth, Universitätsstr. 30, 95447 Bayreuth, Germany
{lukas.sauer, dominik.henrich}@uni-bayreuth.de

Abstract. Using untapped potential in smaller enterprises requires methods for robot programming usable by non-experts. In this paper, we propose an approach based on finite state automata. The basic principle is kinesthetically guiding the robot through tasks and programming state transitions to represent absolute movement, relative movement (to the current pose or to that of an object), or interaction with saved pose variables. The automata structure allows us to represent control flow (loops and branching according to conditions in the perceived scene). We forgo a visual user interface to avoid switching between input devices. This can save time, aid concentration, and reduce the amount of hardware in the workspace. In our approach, the user executes specific key movements while guiding to trigger operations. We assume that users can program small, well-known tasks only seeing the current scene. The approach was evaluated in a user study. The results show that it can be used effectively, but could be improved upon in terms of intuitiveness.

Keywords: Robot programming · Automata · Haptic.

1 Introduction

While robots have a long and successful history in large scale industry, there is both increasing use and still untapped potential in the context of small and medium-sized enterprises or workshops [10], where there are only domain experts readily available, but no expert robot programmers. This necessitates more intuitive forms of robot programming. Automata-based robot programs are promising in this regard, because they can express control flow in a conceptually easy way: in a given situation and with a perceived input, specify what happens next. A drawback is that such programming systems usually use visual representations

* This work has partly been supported by the Deutsche Forschungsgemeinschaft (DFG) under grant agreement He2696/15 INTROP.
This is a preprint of the following chapter: Lukas Sauer and Dominik Henrich, Extended Robot State Automata for Intuitive Robot Programming, published in Advances in Service and Industrial Robotics, edited by Said Zeghloul, Med Amine Laribi, Juan Sandoval, 2021, Springer reproduced with permission of Springer Nature Switzerland AG. The final authenticated version is available online at: https://doi.org/10.1007/978-3-030-75259-0_7

of the automata, and extensive graphical user interfaces (GUIs) to interact with them. However, we assume that there is a subset of tasks that require the expressiveness of automata (compared to e.g. basic playback approaches), but are at the same time simple enough to be programmed just by traversing through the program structure and locally specifying what needs to happen next. If a GUI can be dispensed with in this way, less hardware is required in the robot's workspace and fewer changes between different input modes are necessary for the programmer: there is e.g. no need to switch between mouse and keyboard and kinesthetically guiding the robot.
In our previous work [12], we proposed using finite state automata with an added function mapping states to robot poses. We termed this Robot State Automata (RSA). That model was conceptually easy enough to be used by non-experts, but had deficits in expressiveness: There was no way to express relative movement (e.g. in the body of a loop), to use object pose information (e.g. to grasp objects at their current perceived location), and to save and later use pose information within the flow of a single execution. In this paper, we will describe a new, extended automata variant termed *Extended Robot State Automata* that aims to eliminate these deficits, while retaining the simplicity of control flow of RSA. We outline the programming system we implemented and which uses specific movements executed while guiding to trigger the different programming operations. The results of a user study for evaluation are also presented.

2 Related Work

Automata as a simple concept are used in robotics in varying contexts [1, 4, 6–8, 11]. There is a number of approaches where users can explicitly generate and edit robot programs in the form of (different variants of) automata [2, 13, 14]; however, all of these use graphical editors. Our research, in contrast, is trying to gauge the validity of an approach without a GUI, as motivated in Section 1.
As input modalities, we would ideally like to use haptic interactions wherever possible, becuase combined with kinesthetic guiding, this allows the user to exclusively interact with the system via the robot. We use *haptic* in this context meaning physical manipulation of the robot, as opposed to tactile interaction via e.g. touch-sensitive surfaces. (In the words of [3]: "There is no consensus over the definitions of tactile and haptic interactions.") There is little previous work on using key haptic gestures while guiding to trigger operations [5, 15], and none directly applicable to our approach. This led us to tailor interactions to our specific operations, as detailed in Section 4.

3 Automata Model

Based on finite state automata and the variant in our previous paper [12], we developed a model we refer to as *Extended Robot State Automata* (ERSA).

Definition 1 *An ERSA is given as a tuple* $M = (Q, \Sigma, P, D, \delta, u, q_s)$ *where*

- Q *is the finite set of states,*
- Σ *is the finite input alphabet,*
- P *is the space of robot poses,*
- $D = P^n = P \times \cdots \times P$ *is the pose variable space (for* n *pose variables used)*
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \to Q$ *is the state transition function,*
- $u : Q \times \Sigma \cup \{\varepsilon\} \times P \times D \times P \cup \{\varepsilon\} \to P \times D$ *is the update function, and*
- $q_s \in Q$ *is the initial state.*

Poses $p \in P$ represent transformations: specifically, the current transformation from the world frame to the robot's NSA[1] frame plus information on the tool state for robots, or from the world frame to the object frame for objects. The number n of variables is fixed for any single program. Correspondingly, the variable space D differs in dimension between automata, but can be specified for single automata.
With $\Sigma = \{s_0, s_1, \ldots, s_k\}$, the individual s_i are identifiers for all branch conditions that occur in the program. These consist of one or more pairs of an object identifier[2] and some information about which poses this object is accepted with (e.g. a subspace of P). By s_0 we denote a special branch that represents the case in which no object was found.
The partial function δ determines state transitions as usual, i.e. if an ERSA is in $q \in Q$ and detects a branch $s \in \Sigma$, the automaton will transition to $\delta(q, s)$ if defined. By $\varepsilon \notin \Sigma$ we denote an empty input, i.e. the system transitions from q to $\delta(q, \varepsilon)$ without checking for a specific branch. As ERSA need to be deterministic, if, for a state q, $\delta(q, \varepsilon)$ is defined, no $\delta(q, s)$ must be defined for any $s \in \Sigma$.
The update function u specifies how the state of the robot and its variables changes on a transition (as opposed to δ specifying how the logical state of the program changes). For convenience, we define restrictions $u^{q,\sigma}$ of u to a specific state q and input σ:

$$\begin{aligned} &\text{let } u^{q,\sigma} : P \times D \times P \cup \{\varepsilon\} \to P \times D \\ &\text{s.t. } \forall p_a \in P, d \in D, p_b \in P \cup \{\varepsilon\} : u^{q,\sigma}(p_a, d, p_b) = u(q, \sigma, p_a, d, p_b) \end{aligned}$$

This allows us to specify such restricted update functions at each transition in an automaton. We state that $u^{q,\sigma}$ is defined for all $q \in Q$ and $\sigma \in \Sigma \cup \{\varepsilon\}$ for which $\delta(q, \sigma)$ is defined. These $u^{q,\sigma}$ can take one of five forms:

$$u^{q,\sigma} \in \left\{ \begin{array}{l} (p_a, d, p_b) \mapsto (\bar{p}, d), \\ (p_a, d, p_b) \mapsto (p_a \cdot \bar{p}, d), \\ (p_a, d, p_b) \mapsto (p_b \cdot \bar{p}, d), \\ (p_a, d, p_b) \mapsto (p_a, d_1, \ldots, d_{k-1}, p_a, d_{k+1}, \ldots, d_n), \\ (p_a, d, p_b) \mapsto (d_k, d) \end{array} \right\}, 1 \leq k \leq n, \bar{p} \in P$$

Here, $p_a \in P$ is the current pose, and $d \in D = P^n$ is the current variable vector. $p_b \in P \cup \{\varepsilon\}$ is the object pose from the detected branch, if an object

[1] defined via normal/sliding/approach vectors
[2] Objects need to be recognized. In pratice, we use a database of known objects.

was detected. (For $\sigma = s_0$, the third option is not applicable.) The $u^{q,\sigma}$ map to the new pose that the robot moves to and the updated variable vector. So, the first three options above represent, in order: moving to a new absolute pose $\bar{p}$; moving to a new relative pose (i.e. applying a transformation $\bar{p}$ to the current pose), moving somewhere relative to an object (applying a transformation $\bar{p}$ to the object pose). In all of these, the transformations $\bar{p}$ are constant, and the variable vector remains unchanged. Finally, in the last two options, $u^{q,\sigma}$ can store the current pose as a component of the variable vector d, or move to one of the poses stored in d.[3]

4 Programming System

For programming ERSA as defined above, multiple interactions are necessary. We use kinesthetic guiding, i.e. the robot is moved by hand while compensating for its own weight. To input commands, haptic interactions are favored, since they do not require any additional hardware beside the robot and can be performed while keeping the hands on the robot. Other inputs in the current implementation are via a block of eight buttons mounted on the gripper, which can also be operated while guiding. Finally, the robot has a mounted camera, which could be used to recognize hand gestures. This has not been implemented, but we will point out several places where it would be a convenient extension in the future.

New states can be simply added into Q, but when a new transition is generated (as necessary with new states), the corresponding update function $u^{q,\sigma}$ needs to be specified. We decided to use separate interactions for the different update function options, and have them generate new successor states automatically.

For the first two update options (absolute and relative movement), we use two symmetric haptic interactions. For an absolute move, from a standstill the robot's gripper is moved down (in negative world z direction) a short distance, then up again, to another standstill. The mnemonic here is a pinning gesture (fixating the robot's pose like with a pin on a pinboard). For a relative move, the interaction is vertically flipped (moving the gripper first up, then down again).

Movement to a pose relative to that of an object is generated via one of the buttons. Here, for a potential future gesture recognition, pointing at an object in the scene would be an obvious candidate.

Saving and loading pose variables is triggered by another pair of haptic interactions, twisting the robot's last joint either clockwise and back, or the reverse. This actually generates an update function $u^{q,\sigma}$ where the current pose is saved to or restored from a pose variable in execution (as opposed to saving a pose once in programming). There is no abvious mnemonic for these, but they proved easy to remember.

Since all previous commands add new states by default, we need another one for transitions to existing states. We use a haptic interaction: guiding the gripper

[3] Specifying the index k, i.e. which component of the variable vector to write to or read from on a specific transition, is part of programming.

in a small circle in a horizontal plane (again starting and ending in a standstill). The mnemonic here is that linking to a previous state closes a circle or loop. Marking that the next transition should require an input is done via one of the buttons. The corresponding input s is added to Σ and, in execution, compared to. The transition itself can then be generated by any of the previous commands. In a state that has a transition which requires an input, additional transitions can be generated (via the same command inputs). In that case, any new inputs are also added into Σ for later comparison.

5 Evaluation

Fig. 1. Left: experimental setup. Objects arrive in the black rectangle and have to be either dipped into the green container and put back, or placed at the first free spot marked in black. Right: reminder icons on the robot. Up/down arrows represent the pin gestures for relative/absolute movement. Horizontal circle represents the linking gesture. Right/left arrows on the flange are labeled 'save'/'restore' (for pose variables).

We conducted a user study on the viability of the approach. The setup consisted of a Kuka LWR IV, with a Robotiq gripper as tool and an Intel RealSense D435 RGB-D-camera mounted as an eye-in-hand. Furthermore, a mini-keyboard with eight buttons was attached to the gripper. The buttons are labeled with icons, and another set of icons illustrating the haptic interactions was pasted onto the robot. Both the general setup and these icons are shown in Fig. 1.
Experiments consisted of two parts: an introductory tutorial, and an actual programming task. In the tutorial part, participants were shown all of the interactions (their triggers and their effects). They also had the opportunity to try the haptic interactions. Then, in the programming part, the following task was given: Two types of objects would arrive in a marked area in the workspace. Objects of one of those types had to be picked up, dipped into a container

(symbolizing e.g. a liquid coating), then put back at the exact pickup location. Objects of the other type had to be placed at locations marked by crosses, at the first unoccupied position starting from the right.

This task, illustrated in Fig. 2, uses all ERSA capabilities: branching according to the perceived scene; picking up objects at their current pose; saving and restoring a precise, but a priori unknown (pickup) location; relative movement from one marked location to the next; and loops within the automaton structure.

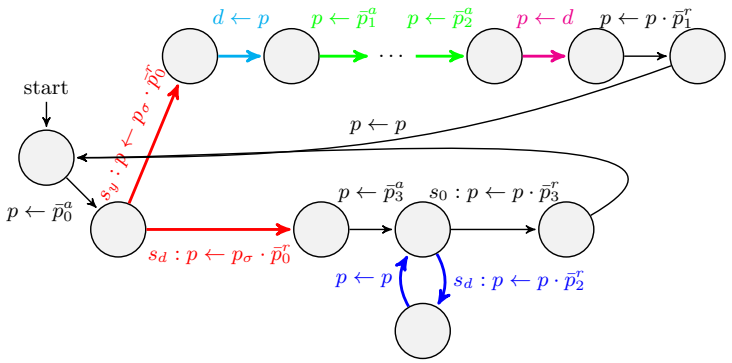

Fig. 2. A simpflified automaton for the task. Edges labeled "σ:" execute only on detection of that branch. As the example only uses one pose variable, d is used in place of d_1. Updates are specified in a pseudocode assignment fashion; e.g. "$d \leftarrow p$" means saving the current pose in the pose variable (everything else is unchanged).
Uses movement relative to the object pose for pickup (red). In the top path, saves the pose (cyan), dips the object (absolute moves, green), then restores the pose (magenta). In the bottom path, moves to the next target location in a loop (relative moves, blue).

To evaluate effectiveness and intuitiveness of the programming process, we used several questionnaires collated in the MINERIC toolkit [9]. Specifically, participants were asked to gauge the complexity value (COM) of the system (once after a brief general introduction, but before the tutorial, and once after programming), the mental effort of programming (SSEE), and to answer a questionnaire on different aspects of their programming experience (QUESI), while the researcher noted the degree of success in programming the task (PAC-U).

Experiments were conducted in October 2020. Due to Corona pandemic-related restrictions, only a small sample size of seven participants was taken. The results are presented in the diagrams of Fig. 3. There are two main considerations: effectiveness and intuitiveness of programming. As for effectiveness, the PAC-U mean value of 2.43 out of 5 corresponds to a mostly successful programming process with frequent questions posed at or hints requested from the researcher. The G subscore mean of 3.95 expresses that the QUESI-statements on the achievement of goals with the programming system were generally agreed with. As for intuitiveness, results are less satisfying. The COM-C mean (overall complexity score)

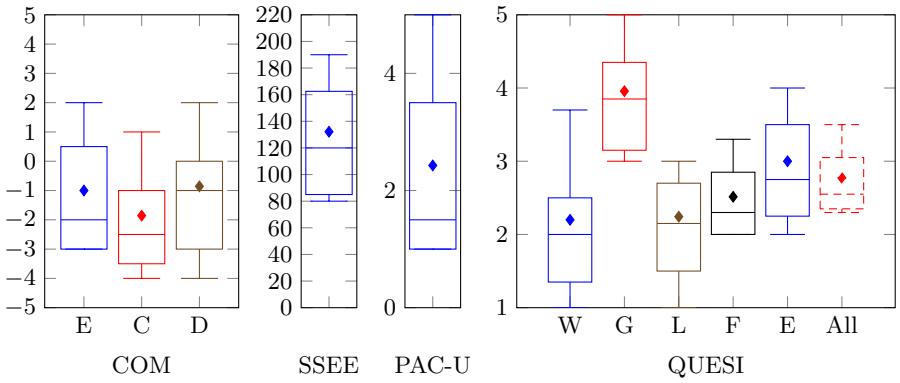

Fig. 3. Left to right: perceived complexity of the system before/after use and difference (negative is more complex), mental effort (lower is better), effectiveness of programming (higher is better), and agreement with positive statements in QUESI (higher is better). The mean PAC-U and G values of 2.43 and 3.95 respectively show that participants were able to program using the system (albeit requesting hints multiple times).

is –1.86 and the COM-D mean (difference to expected complexity) is –0.86, both on a scale from 5 to –5. This means that participants found the system both slightly complicated to use, and slightly more complex than they expected from the brief introduction. The SSEE mean of 132.14 out of 220 depicts a mental effort between 'rather much effort' and 'great effort'. Finally, the W, L and F QUESI subscores are all below the neutral value of 3, meaning participants tended not to agree with the positive statements on mental workload, low effort of learning, and familiarity with the system.

6 Conclusion

In this paper we presented Extended Robot State Automata (ERSA), a substantial extension of our previous automata model for robot programming that eliminates several deficits identified in the original model. We outlined our system for programming ERSA, still without using a GUI. We conducted a user study to evaluate the effectiveness and intuitiveness of the approach. The former is positive, while the latter can still be improved upon. Overall, this serves as a proof of concept that the ERSA model can be used for programming, in particular without a GUI even for moderately complex tasks.
In future research, intuitiveness of the programming system could be improved. For instance it frequently led to errors in the study that pose variables need to be saved at a point before they need to be used, which participants tended to forget. Gestic interactions are an obvious extension to potentially get rid of the handful of button interactions. This would necessitate a direct comparison to evaluate. Finally, the system could generate some of the underlying automata structure

without explicit programming. This applies e.g. to loop structures, where participants struggled deciding which states to link. Instead, they could demonstrate the first and second execution of the loop body, and the system could roll those into a loop according to correspondence of the states and transitions.

References

1. Brooks, R.A.: A Robust Layered Control System For A Mobile Robot. IEEE J. Robotics and Automation **2**(1), 14-–23 (1986).
2. Brunner, S.G., Steinmetz, F., Belder, R., Dömel, A.: RAFCON: A graphical tool for engineering complex, robotic tasks. In: Proc. 2016 IEEE IROS, pp. 3283—3290. IEEE (2016). https://doi.org/10.1109/IROS.2016.7759506.
3. Carter, J., Fourney, D.: Research Based Tactile and Haptic Interaction Guidelines. In: Guidelines on Tactile and Haptic Interaction, pp. 84—92 (2005).
4. Grollman, D.H., Jenkins, O.C.: Can we learn finite state machine robot controllers from interactive demonstration? In: Sigaud O., Peters J. (eds.) From Motor Learning to Interaction Learning in Robots. SCI, vol 264. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-05181-4_17.
5. Jokinen, K., Wilcock, G.: Multimodal Open-Domain Conversations with the Nao Robot. In: Mariani J., Rosset S., Garnier-Rizet M., Devillers L. (eds.) Natural Interaction with Robots, Knowbots and Smartphones. Springer, New York (2014). https://doi.org/10.1007/978-1-4614-8280-2_19.
6. König, L., Mostaghim, S., Schmeck, H.: Decentralized Evolution of Robotic Behavior Using Finite State Machines. Int. J. Intelligent Computing and Cybernetics **2**(4), 695-–723 (2009). https://doi.org/10.1108/17563780911005845.
7. Marino, A., Parker, L., Antonelli, G., Caccavale, F.: Behavioral control for multi-robot perimeter patrol: A finite state automata approach. In: Proc. 2009 IEEE ICRA, pp. 831–836. IEEE (2009). https://doi.org/10.1109/ROBOT.2009.5152710.
8. Orendt, E.M., Henrich, D.: Control flow for robust one-shot robot programming using entity-based resources. In: Proc. 18th ICAR, pp. 68–74. IEEE (2017). https://doi.org/10.1109/ICAR.2017.8023498.
9. Orendt, E.M., Fichtner, M., Henrich, D.: MINERIC toolkit: Measuring instruments to evaluate robustness and intuitiveness of robot programming concepts. In: Proc. 26th IEEE RO-MAN, pp. 1379–1386. IEEE (2017). https://doi.org/10.1109/ROMAN.2017.8172484.
10. Perzylo, A., et al.: SMErobotics: Smart robots for flexible manufacturing. IEEE RAM **26**(1), 78-–90 (2019). https://doi.org/10.1109/MRA.2018.2879747.
11. Riano, L., Mcginnity, T.M.: Automatically composing and parameterizing skills by evolving finite state automata. RAS **60**(4), 639–650 (2011). https://doi.org/10.1016/j.robot.2012.01.002.
12. Sauer, L., Henrich, D., Martens, W.: Towards Intuitive Robot Programming Using Finite State Automata. In: Benzmüller C., Stuckenschmidt H. (eds.) KI 2019, LNCS, vol. 11793. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30179-8_25.
13. Steinmetz, F., Wollschlager, A., Weitschat, R.: RAZER—A HRI for Visual Task-Level Programming and Intuitive Skill Parameterization. IEEE RAL **3**(3), 1362–1369 (2018). https://doi.org/10.1109/LRA.2018.2798300.

14. Thomas, U., Hirzinger, G., Rumpe, B., Schulze, C., Wortmann, A.: A new skill based robot programming language using UML/P Statecharts. In: Proc. 2013 IEEE ICRA, pp. 461–466. IEEE, (2013). https://doi.org/10.1109/ICRA.2013.6630615.
15. Wösch, T., Feiten, W.: Reactive motion control for human-robot tactile interaction. In: Proc. 2002 IEEE ICRA, vol. 4, pp. 3807–3812. IEEE (2002). https://doi.org/10.1109/robot.2002.1014313.