Comparing Motion Generation and Motion Recall for Everyday Mobile Manipulation Tasks

Carmen Lopera, Hilario Tomé, Adolfo Rodríguez Tsouroukdissian
PAL Robotics S.L., Carrer de Pujades 77-79, 08005 Barcelona, Spain

Freek Stulp
Cognitive Robotics, École Nationale Supérieure de Techniques Avancées, 32 Boulevard Victor, 75015 Paris, France
FLOWERS team, INRIA Bordeaux Sud-Ouest, Talence, France

I. Introduction

When first posed with the problem 15×15, we may *generate* the answer by applying a set of rules, e.g breaking the problem down into $(10 + 5) \times 15$ and solving the subcomponents of these simpler multiplications first [2]. But after having solved this problem several times, we simply *recall* that the answer to 15×15 is 225. This distinction between *generation* and *recall* can also be applied to motor planning [2], as described in the next two sections.

A. Motion Generation

Motion generation is based on planning with internal models, to determine which movements should be used to achieve a specific goal [2]. An example in robotics is Rapidly-exploring Random Trees (RRTs) [6], where a (bidirectional) search based on an internal model of the environment and the robot is used to find a collision-free motion plan from the current state to the goal state. The main advantage of using motion generation is its generality – in theory, if a motion plan connecting two states exists, it can be found. We believe this generality is one of the main reasons for the popularity of sampling-based motion planning in state-of-the-art mobile manipulation. A disadvantage, however, is that accurate internal models must be available to perform the search. Also, changes in dynamic environments might invalidate the plan, and thus require re-planning.

B. Motion Recall

In motion recall, a specific movement is stored, e.g. a particular movement for grasping objects. When the appropriate task context arises, e.g. we see a cup on a table, we recall this movement and execute it. This is analogous the remembering that the number 225 is the answer to 15×15, rather than generating it from scratch. Small adaptations may be necessary to adapt the stored movement to the specific task parameters, e.g. the specific position of the object on the table.

Recalling motion plans exploits the fact that in almost all activities of daily living, related tasks are encountered over and over again. Therefore, humans "tend to solve similar or even identical instances over and over, so we can keep recycling old solutions with minor modifications" [4]. An example of this approach in robotics is using Dynamic Movement Primitives [5], which represent a goal-directed movement as a set of dynamical systems equations, where the goal parameters may be adapted to the specific task. Motion primitives are typically initialized with imitation learning, and then 'replayed' in similar task contexts. Some advantages of motion recall with motion primitives are: its negligible computational cost; reproducibility of the movement, since the same primitive is executed in the same context; on-line adaptation without replanning, e.g. dynamic obstacle avoidance. A disadvantage is that each motion primitive can only be applied to a limited task context, and is thus not very general.

C. Project Goals

The main goal of our collaboration is to quantitatively evaluate the advantages of using motion generation and/or motion recall for everyday manipulation tasks on a commercial robot platform. We do so along the dimensions of generality, motion variability, and computational load.

II. Implementation

Our evaluation is performed with the REEM humanoid robot developed at PAL Robotics, and depicted in Fig. 1. To implement *motion generation*, we use the open-source implementation LazyRRT in the OMPL library [11], combined with the constraint aware spline smoother [9]. For *motion recall*, we use an open source implementation of Dynamic Movement Primitives [10] to represent, store and execute motion plans.

III. Results

The results with respect to the tree evaluation measures are depicted in the video, as well as Fig. 1.

- **Motion Variability.** Because a new plan is generated from scratch with LazyRRT each time, there is high variance in the generated movement. In the video this is visualized by overlaying 4 executions. In Fig. 1 the mean and standard deviation over 100 trials is shown. When using Dynamic Movement Primitives, the same primitive is executed at each trial, so the variance is very low – in the video, it is hardly visible that 4 trials have been overlayed.

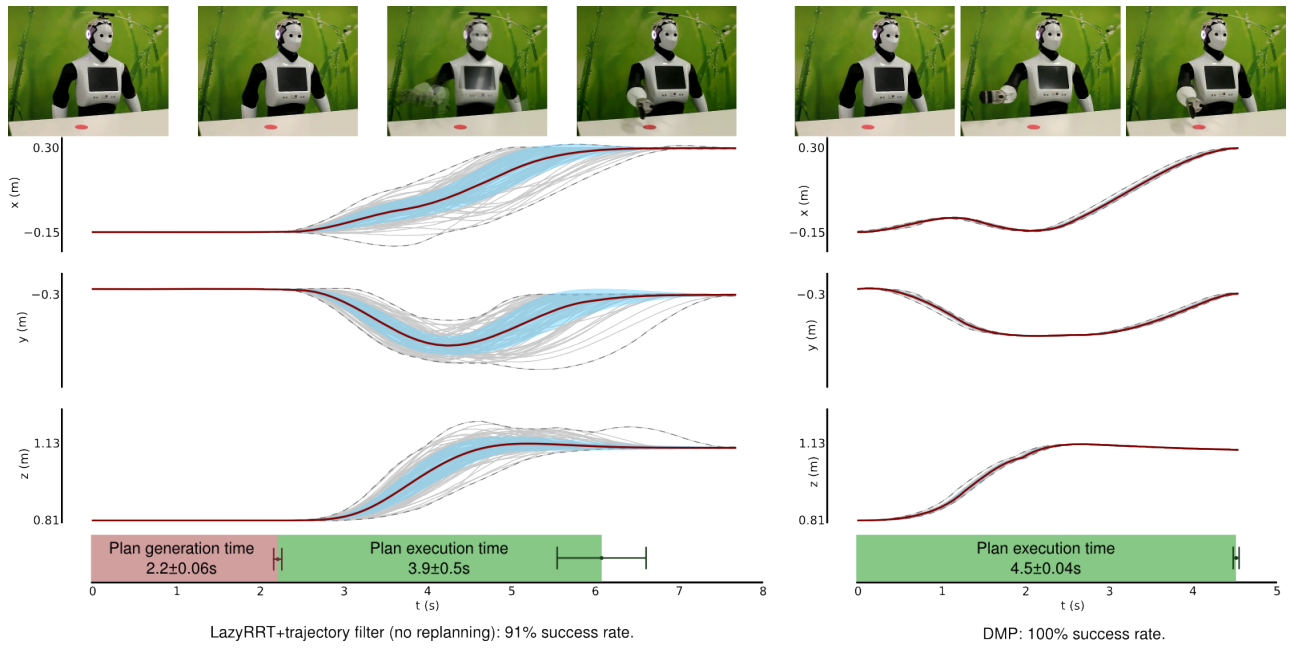

Fig. 1. Evaluation of the variability and planning time of LazyRRT (left) and DMPs (right).

- **Computational Load.** First of all, the planning time for the LazyRRT makes up 35% of the total execution time, averaged over 100 trials. For the Dynamic Movement Primitives, this is 0%.
- **Generality.** The LazyRRT is a probabilistically complete motion planner. If given enough planning time, it will eventually find a solution to the problem at hand, if it exists. On the other hand, Dynamic Movement Primitives perform well within their prescribed context, but poorly outside of it. In our experiments, the task context is limited to reaching different positions over a moderately tall table. Within the table plane, the motion primitive is able to generalize to targets more than 0.4m away from the learned goal, but when faced with a table that is lower by 0.25m, the motion primitive fails to render the qualitatively different motion that is required.

IV. Conclusion and Outlook

Our preliminary results show that the advantages of motion generation and motion recall are complementary, which suggests a system that uses both strategies to have the best of both worlds. This comparison therefore constitutes the first step in a broader research project, in which we will investigate how methods based on motion recall for motion planning can be integrated for general, efficient, and reproducible behavior in mobile manipulation. Some of the specific research questions we seek to address are:

- How can we initialize motion primitives with the output of sampling-based motion planners [1]? I.e. how can we *recall* plans that have previously been *generated* [8]?
- How can robot (learn to) recognize in which task contexts a motion primitive can be successfully executed [3]?
- How repetitive are everyday manipulation tasks, and how many motion primitives are required to solve the bulk of them?
- How can we plan with motion primitives, for instance to generate sequence of motion primitives for more complex pick-and-place tasks?
- How do optimization principles in sampling-based motion planning [7] influence the variability of the motion plans they generate?

References

[1] Dmitry Berenson, Pieter Abbeel, , and Ken Goldberg. A robot path planning framework that learns from experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[2] R. G. Cohen and D. A. Rosenbaum. Where grasps are made reveals how grasps are planned: generation and recall of motor plans. *Exp Brain Res*, 157(4):486–495, 2004.

[3] Debadeepta Dey, Tian Yu Liu, Martial Hebert, and J. Andrew Bagnell. Contextual sequence prediction with application to control library optimization. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[4] Ian D. Horswill. *Specialization of perceptual processes*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1993.

[5] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[6] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[7] A. Perez, S. Karaman, M. Walter, A. Shkolnik, E. Frazzoli, and S. Teller. Asymptotically-optimal manipulation planning using incremental sampling-based algorithms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[8] Martin Stolle and Christopher G. Atkeson. Finding and transferring policies using stored behaviors. *Auton. Robots*, 29(2):169–200, 2010.

[9] `http://www.ros.org/wiki/constraint_aware_spline_smoother`.

[10] `https://bitbucket.org/usc_clmc/usc-clmc-ros-pkg/src/94ca0d5afaa0/dmp`.

[11] `http://ompl.kavrakilab.org`.