Robot-driven Trajectory Improvement for Feeding Tasks

Travers Rhodes[1] and Manuela Veloso[2]

Abstract— **Kinesthetic learning is a type of learning from demonstration in which the teacher manually moves the robot through the demonstrated trajectory. It shows great promise in the area of assistive robotics since it enables a caretaker who is not an expert in computer programming to communicate a novel task to an assistive robot. However, the trajectory the caretaker demonstrates to solve the task may not be optimal. The demonstrated trajectory could be suboptimal because the teacher doesn't know the optimal trajectory for the robot, or because the teacher has difficulty moving all the robotic joints precisely along the desired trajectories. We propose the Parameterized Similar Path Search (PSPS) algorithm to extend kinesthetic learning so that a robot can improve the learned trajectory over a known cost function. This algorithm is based on active learning from the robot through collaboration between the robot's knowledge of the cost function and the caretaker's knowledge of the constraints of the assigned task.**

I. Introduction

In the US, about three in every 1000 children have cerebral palsy [6]. Since at least 1963, when James Reswick developed the Case Research Arm Aid—Mark I, researchers have been trying to create robotic aids to assist in everyday activities for people living with disabilities [17]. In 1987, Mike Topping began work on HANDY-1, a robot designed to help people with cerebral palsy at mealtimes, and his product was eventually commercialized [21], [22]. Compared to a caretaker, a robot feeding system like HANDY-1 allows the user to have more control over the choice of food for each bite, allows the user to choose the pace of the meal, allows for more consistent feeding, and makes the meal more engaging for the user [21].

There are currently a number of commercial robotic products available which assist in the feeding task, including Bestic [5], Meal Buddy [15], Mealtime Partner [10], My Spoon [18], and Obi [11]. For these products, a human agent (generally a caretaker) sets up the robot so that its feeding trajectory reaches the appropriate feeding location for the user. For example, for the Obi, the caretaker kinesthetically shows the robot the target feeding location by holding down a button and then manually moving the spoon to the desired feeding position. The Obi then computes a trajectory from the food to that target location [11].

Our research aims to improve the capabilities of autonomous feeding robots to allow a caretaker to specify the full trajectory of the robotic arm rather than just the final position. For example, certain foods with less friction, like candy-coated chocolates, may require slower trajectories than stickier foods, like fried rice. Specifying the full feeding trajectory could also enable a caretaker to expand the capabilities of the feeding robot to perform additional tasks, like combining foods before each bite [20] or dipping each bite in sauce before moving the food to the feeding location.

Kinesthetic learning (or, more broadly, any form of learning from demonstration) is a language-agnostic [7], efficient [7], and intuitive [3] teaching method. Most importantly, it could allow a person who is not an expert in computer programming to communicate a novel task to a robotic agent [2].

However, while a kinematic demonstration gives an example solution to the new task, the demonstrated trajectory can be suboptimal [3]. For many cost functions, like minimizing the acceleration over a fixed time range, we do not expect the teacher to know the optimal trajectory. Even if the teacher knows the optimal trajectory, it can be challenging during a demonstration for the teacher to move all the robot's joints simultaneously along the optimal trajectory. Therefore, even though the human teacher has the most domain knowledge about what constitutes successful completion of the task, we expect the robot to nevertheless be able to improve on the trajectory taught by the teacher.

We contribute the Parameterized Similar Path Search (PSPS) algorithm, which allows the robot to improve motions learned from kinesthetic learning. We apply PSPS to feeding-adjacent tasks and contribute our finding that for our example task it is not necessary for the teacher to give specific guidance on when a suggested trajectory failed in order to see improvements to the learned trajectory from PSPS.

II. Related Work

There are two main areas of related work: related work involving feeding-assistance robots and related work involving learning from demonstration.

A. Robotic Feeding Assistants

In addition to commercial robotic feeding assistants [5], [10], [11], [15], [18], there are currently a number of ongoing research projects to improve robotic feeding assistants, including research into adding detection for the position of the user's mouth and whether the mouth is open [14], into using a general purpose manipulator (eg: the PR2 robot) for the feeding task rather than a specialized robot [13], into anomaly detection to detect errors during the feeding task [12], and into the optimal robot design for culture-specific food [20].

Rather than focusing on improving a particular feeding action, or focusing on the design of the feeding robot,

[1]Travers Rhodes is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. `traversr@andrew.cmu.edu`

[2]Manuela Veloso is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. `veloso@cs.cmu.edu`

Preprint submitted to 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems. Received March 1, 2018.

we focus on expanding the flexibility of feeding robots by introducing our PSPS algorithm so that the flexibility of kinesthetic learning can be more optimally applied to the robotic feeding assistant domain.

Another current area of research in robotic feeding is in shared autonomy, where the user and the robot together direct the robotic arm during the feeding task [9]. Our work focuses on optimizing trajectories for autonomous robots.

B. *Learning from Demonstration*

The idea of kinesthetic learning for assistive robotics is not new. The Case Research Arm Aid—Mark I had kinesthetically programmed motions included, though latency in retrieving the programmed motions from the expensive magnetic tape system made it impractical [17]. We hope that our PSPS algorithm will encourage continued adoption of kinesthetic learning for assistive robotics.

Work in learning from demonstration often focuses on the robot finding a successful trajectory, like the seminal research in [4]. Our work is different because, unlike in [4], replaying our training trajectory led to sucessful task completion. We believe this is partly due to the fact that our setup did not have correspondence issues, which can occur when the robot maps observed demonstrations to its own body. We avoid correspondence issues by demonstrating the trajectory directly on the robot. Since replaying our training trajectory was successful, our work focuses instead on the robot learning to *improve* a successful trajectory.

Our learning from demonstration paradigm is similar to Argall et al.'s work in [2] in that PSPS can consider feedback from the teacher on what part of the trajectory was incorrect. We find, however, that PSPS performs well without this temporal feedback on the time of an error. The robot has additional knowledge of what it is trying to optimize in the PSPS algorithm since it knows the explicit cost function, and so the robot is able to perform this optimization even without direct guidance from the teacher on how to do the search. This combination of a robotic understanding of a cost function with kinesthetic learning also appears in [19], but we differentiate our work in improving the learned kinesthetic motion itself, rather than on when or how to apply that motion.

Recent work has looked at the possibility of using neural networks as a means of learning from a single demonstration [7]. For explainability and the ability to understand mathematical bounds on robotic actions, we chose not to follow this neural network approach.

In [23], Ye and Alterovitz extend kinesthetic learning in a similar home service domain. Their paper focuses on learning task constraints so that the robot can succeed at the task in new scenarios, for example, after the addition of obstacles. Instead of attempting to generalize learned motions, our work focuses on improving the particular learned kinesthetic motions in the given domain so that they are better than the taught trajectory.

III. Problem Statement and Formalization

We address the problem of how an assistive robot can improve on a trajectory learned through kinesthetic teaching. The robot is kinesthetically trained on some task, as in Fig. 1a, and our problem is to construct a learning algorithm whereby the robot can improve on that task, as in Fig. 1b.

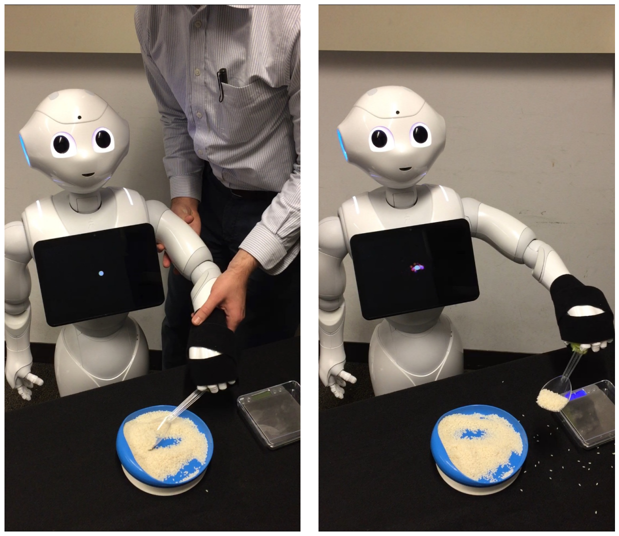
(a) Pepper learning from kinesthetic training (b) Pepper improving learned trajectory using PSPS

Fig. 1: Images of the Pepper robot learning the rice serving task presented in Section V

For an assistive feeding robot with m degrees of freedom, let $\mathcal{J}$ be the set of valid static configurations $\mathcal{J} = \{(j_1, j_2, \ldots, j_m) \in \mathbb{R}^m\}$ where j_i is the parameterization of the i^{th} degree of freedom. For example, if the i^{th} degree of freedom corresponds to a revolute joint, then j_i would measure the angle of that joint. We consider a trajectory to be a set of pairs of joint states and times $\{(\boldsymbol{j}_1, t_1), \ldots, (\boldsymbol{j}_k, t_k)\}$, with $\boldsymbol{j}_i \in \mathcal{J}$, $t_i \in \mathbb{R}$, and $t_i < t_{i+1}$. We call the set of all trajectories Φ. Since our robot is trained kinesthetically, the training demonstration is an element of the set Φ.

We assume there exists some cost function $C : \Phi \to \mathbb{R}$, known to the robot, that gives a measure of the cost of any trajectory. This cost function could be any cost, like the total path length, the smoothness of the path, or the energy needed to achieve that trajectory.

We assume that the teacher knows some task-success-evaluation function $S : \Phi \to \{0, 1\}$, unknown to the robot, that returns 1 if the input trajectory satisfactorily completes the task and 0 otherwise. The task-success-evaluation function is evaluated by the teacher, and so the teacher decides whether the robot's attempt to complete the task was successful. For example, if the task is to scoop up a dumpling, dip it in sauce, and move it to the feeding location, then the success function S will only return true if the trajectory accomplishes all those tasks to the satisfaction of the teacher. The robot is able to ask the teacher to evaluate the success function $S(\phi)$ for any trajectory ϕ. While we allow the robot to ask the teacher to evaluate any trajectory, that trajectory evaluation does involve the robot actually executing the trajectory. Since novel exploratory trajectories can be dangerous to execute on a real robot, it's important that the robot gradually change and improve its trajectory around a known viable trajectory rather than searching across the whole space of trajectories.

Each time the robot asks the teacher to evaluate the success function is a learning step. Pursuant to [2], we consider enhancing this learning problem by having the teacher also provide an estimate of the time(s) when the robot trajectory failed the task, though our experiments find satisfactory results without including that component.

We define our problem as follows: The robot is demonstrated a single successful trajectory $\phi_d \in \Phi$. The robot asks the teacher to evaluate the success of N different trajectories. The robot's learning problem is to find, after these learning steps, a learned trajectory ϕ_l for which $S(\phi_l) = 1$ and which makes $C(\phi_d) - C(\phi_l)$ as large as possible.

We note that this formalization can be extended naturally to handle a stochastic success function S_s, where $S_s(\phi)$ returns 1 with probability $p(\phi)$ and 0 with probability $1-p(\phi)$, by having the robot attempt to maximize $\mathbf{E}[S_s(\phi_l)(C(\phi_d) - C(\phi_l))]$. This maximization can be empirically estimated by evaluating $S_s(\phi_l)$ multiple times.

IV. Parameterized Similar Path Search (PSPS)

The PSPS algorithm is based on the assumption that similar trajectories are likely to have similar success values. Thus, we want the algorithm to focus its search on trajectories that are similar to the training trajectory. We also note that, because exploratory trajectories on a real robot can be dangerous, having the learning algorithm focus on trajectories similar to the kinesthetically trained trajectory is also important from a safety perspective.

We therefore define the PSPS algorithm as follows. First, the robot defines a function $d : \Phi \rightarrow \mathbb{R} \geq 0$ that measures how different a new trajectory is from the training trajectory. Second, the robot picks some distance parameter $\delta_{cur} \in \mathbb{R}$ and searches over the set of trajectories $\{\phi \mid d(\phi) < \delta_{cur}\}$ for the trajectory in that set that minimizes the cost $C(\phi)$. The robot then performs a learning step and asks the teacher if that trajectory is successful. If the trajectory is successful, the robot increases δ_{cur} and repeats. If the trajectory is not successful, the robot decreases δ_{cur} and repeats.

If the robot is given information about particular parts of the trajectory that led to failure, the robot may also update the distance function d to make the trajectory more closely align with the training trajectory for those failing timesteps.

This active learning algorithm focuses the robot's search on trajectories that improve the cost function C. We expect this algorithm to be robust to the choice of the initial δ_{cur}, as it performs an exponential search in the δ_{cur} parameter. For safety reasons, to avoid the robot executing dangerous trajectories, we suggest that δ_{cur} be initialized to a small value and that the feasibility of the initial δ_{cur} be tested in simulation before this algorithm is run on a new robotic system.

Algorithm 1 Parameterized Similar Path Search Algorithm

```
1: Initialize $\phi_{current\_best}$ to the trained trajectory
2: Initialize $\delta_{min} \leftarrow 0$
3: Initialize $\delta_{cur}$ to some arbitrary value
4: Initialize *isDeltaMaxKnown* $\leftarrow$ False
5: iteration $\leftarrow 0$
6: **while** iteration $< N$ **do**
7:     **if** *isDeltaMaxKnown* **then**
8:         $\delta_{cur} \leftarrow (\delta_{min} + \delta_{max})/2$
9:     **else**
10:        $\delta_{cur} \leftarrow \delta_{cur} * 2$
11:    **end if**
12:    Define similar trajectories $\Phi_\delta = \{\phi \mid d(\phi) < \delta_{cur}\}$
13:    $\phi_{test} \leftarrow \arg\min_{\phi \in \Phi_\delta} C(\phi)$
14:    Ask teacher to evaluate Success $\leftarrow S(\phi_{test})$
15:    **if** Success **then**
16:        $\phi_{current\_best} \leftarrow \phi_{test}$
17:        $\delta_{min} \leftarrow \delta_{cur}$
18:    **else**
19:        $\delta_{max} \leftarrow \delta_{cur}$
20:        *isDeltaMaxKnown* $\leftarrow$ True
21:    **end if**
22:    **if** The robot receives additional feedback **then**
23:        The robot updates $d$ based on that feedback
24:    **end if**
25: **end while**
26: **return** $\phi_{current\_best}$
```

A perfect distance function d would be one where for some value $\delta_{success}$ we would have $d(\phi) < \delta_{success} \implies S(\phi) = 1$ and $d(\phi) > \delta_{success} \implies S(\phi) = 0$. While it would be idealistic to imagine we could find such a perfect distance function, it is nevertheless helpful in guiding the search for a good distance function.

V. Experimental Evaluation and Results

We run experiments to better understand the feasibility of this algorithm and its real-world performance. We are interested in evaluating how much this algorithm is able to optimize sample learned trajectories and whether it is necessary for the teacher to provide information on the time of trajectory failure (lines 22-24 in Algorithm 1) to make the PSPS algorithm effective.

We ran the PSPS algorithm in both the 2D simulated feeding world and on a real robotic rice serving task.

A. 2D Feeding Task Simulation Setup

We first test this algorithm in simulation in a low-dimensional environment so it is easier to visualize its behavior. We consider a two-dimensional world where the robot state J is the x, y coordinates of a spoon. Inspired by our feeding problem, the task constraints are to generate a trajectory that passes from the start location to inside the rice outlined in blue in the bowl on the right (to pick up the food) and then to the green area in the bowl on the left (to deposit the food) without hitting the table or sides of the bowls. The locations of the bowls and table in this simulation, along with the training trajectory curve demonstrated by human input through a mouse, are displayed in Fig. 2.

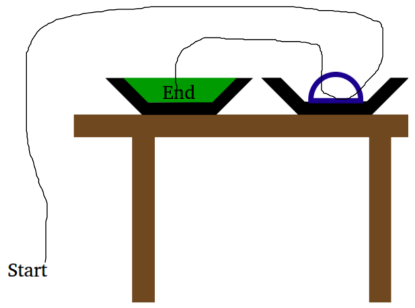

Fig. 2: The feeding task trajectory input by the human teacher, going from start to rice bowl (blue outline) to target bowl area (green)

In this simulation, we set the goal to minimize the roughness (sum of squared second derivative) of the trajectory, while ensuring that the weighted summed square distance between the new path and the old path over all timestamps is less than the constraint distance. This choice of smoothness function combined with the distance function is known [8] to be solved by natural cubic splines, and we are able to use the MATLAB `spaps` function [16] at each step of the PSPS algorithm to optimize the cost function given the constraint.

In this simulation, the mechanics of the PSPS algorithm interaction between teacher and simulated robot is as follows. The simulated robot performs the learning optimization step of the PSPS algorithm by computing a similar spline trajectory to the training trajectory within some constraint distance δ. Then, the simulated robot draws that new trajectory on the screen for the evaluation of the teacher. The teacher inspects that trajectory and reports back whether the trajectory meets the teacher's task constraints or not.

B. 2D Feeding Task Simulation Results

For convenience in interpretability, we reparameterize our distance constraint to be that the sum of square distances between training and tested paths must be no more than $\delta^2 n$ where n is 544, the number of points in the trajectory. The map giving the layout of the table and bowls is 500x800 pixels, and we begin with $\delta = 5$ pixels. We constrain the spoon trajectory to match the start and ending locations closely by multiplying the start and end-point square errors by 1000 before including those errors in the constraint sum.

Fig. 3 shows the result of running PSPS in the 2D feeding simulation. The training trajectory (here trained by drawing with a mouse on the image) is shown as a dotted black line. The simulated robotic spoon first tries a similar trajectory using PSPS and $\delta = 5$ pixels. Since that trajectory gets marked as successful by the teacher, we plot it in green in Fig. 3. Since that trajectory was successful, the learning agent then increases δ and iterates, following PSPS. We plot the tested trajectories in Fig. 3, with green showing successful trajectories and red showing failures. After six learning iterations, the best trajectory is plotted in blue.

From this experiment we observe that this algorithm can indeed optimize over the training trajectory while satisfying the trajectory constraints without the additional necessity of adding waypoint constraints along the path of the trajectory. For this example case, we find that it is not critical for the PSPS algorithm to receive information about the timing of failures in order to nevertheless find an improved trajectory over the training trajectory.

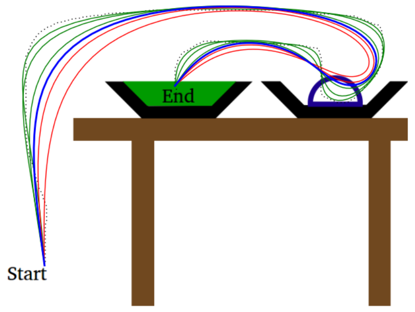

Fig. 3: Spoon trajectories attempted for this task. Red indicates the human teacher marked that attempt as a failure, green indicates success. The learned trajectory after six iterations is in blue. The black dotted line is the demonstration.

C. Pepper Robot Rice Serving Task Setup

We also explore the usefulness of this algorithm on a real robot for a realistic robotic-feeding-related task. We use the Pepper robot from SoftBank/Aldebaran Robotics and attach a spoon to its left hand, The joint configuration of Pepper is shown in Fig. 4.

We task Pepper with scooping up rice from a blue container and dropping the scoop of rice in a target bowl. For this task, we assume that there is a separate process that ensures that there is rice ready to be scooped. Future work may include learning preparatory scraping motions with the spoon to position the rice so that it can be easily scooped, and may include incorporating vision to find where the rice is or stochasticity to ensure that scoops are distributed across different parts of the plate. When vision is incorporated in future work, we hope to experiment with augmenting the learning algorithm to detect the type of food and to learn different trajectories for different types of food. For now, for this task, our robot runs blind (no visual input) and we reset the bowl to be equally full of rice between each attempt so that repeated scoops of rice from the same part of the bowl are equally successful.

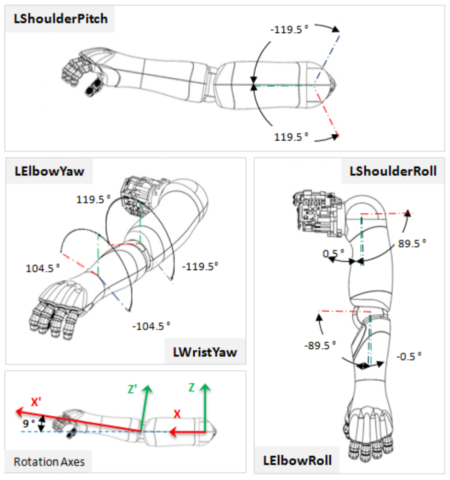

Fig. 4: The arrangement of joints defining the joint space of the left arm of Pepper, courtesy of Aldebaran Robotics [1]

We kinesthetically demonstrate a single serving trajectory for the robot that travels from the starting position to the bowl with rice in it, that scoops up some rice, and that deposits the rice in the target bowl. We weigh the amount of rice deposited in the target bowl after execution, and the trajectory is considered to be a failure by the teacher if the test trajectory deposits less than 50% of the rice that was deposited in the bowl by the training trajectory. In our experiment, the training trajectory deposited 29.8 grams of rice onto the target bowl, so whenever the robot asked us to judge a trajectory, we deemed the trajectory a failure if it deposited less than 14.9 grams of rice in the target bowl. Fig. 1a shows the Pepper robot being taught a sample serving trajectory kinesthetically and Fig. 1b shows the Pepper robot learning improvements to the training trajectory.

We use the same cost/constraint pair that we used in the 2D feeding simulation, with the cost function minimizing the roughness (sum of squared second derivative) of the trajectory, and the distance between two trajectories being measured by the sum of the square distances between them. We measure both the roughness of trajectories and the distance between trajectories in the joint space of the robotic arm. We use the left arm of the Pepper robot, and the joint configuration is displayed in Fig. 4.

D. Pepper Robot Rice Serving Task Results

For convenience in interpretability, we again reparameterize our distance constraint to be that the sum of squared differences must be below $\delta^2 n$, where here n is 91, the number of waypoints in the training trajectory. We initialize δ in the robot feeding task to 0.1 radians, after confirming in a simulated robotic environment that that would not lead to dangerous robotic trajectories. After six iterations of the PSPS algorithm, the robot had bounded δ to between 0.15 and 0.15625 radians. A plot of the amount of rice deposited in the destination bowl as a function of the δ value used in trajectory calculation is given in Fig. 5.

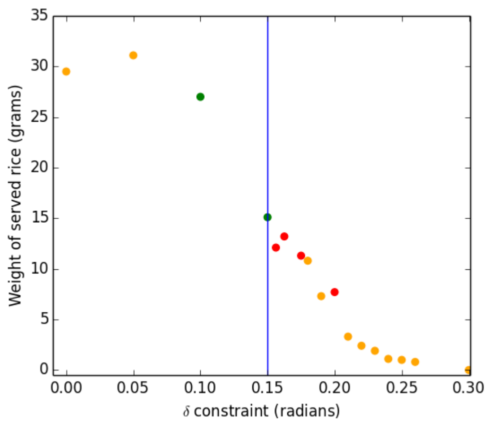

Fig. 5: The weight of rice transferred to the target bowl at each learning step in the PSPS algorithm, parameterized by the distance constraint parameter δ. Red points are those marked by the teacher as failures. Green points succeeded. Orange points show tests that were run outside of the PSPS algorithm to confirm that the PSPS algorithm had found the correct optimal δ value of 0.15 radians (vertical line)

From this plot, we find that the geometric search for the correct δ performed by the robot during the PSPS algorithm is effective. We find that for this example domain, the data follow the expected trend that trajectories that are above a certain difference from the training trajectory will tend to fail to satisfy the teacher's task constraints.

To compute how much PSPS was able to optimize the trained trajectory, we compute, using finite differences, the total sum of squared second derivatives (the cost function) for the training trajectory and the solution found by PSPS. The training trajectory had a total cost of 9.78 radians/second^2. PSPS's learned trajectory had a total cost of 0.461 radians/second^2, showing a marked improvement over the cost function. To visualize the improvement of the PSPS algorithm's result over the roughness cost, we plot the learned ($\delta = 0.15$) trajectory for each joint in Fig. 6. We see from Fig. 6 that the the PSPS algorithm has indeed been able to improve the smoothness of the joint trajectories, while, as noted in Fig. 5, still satisfying the task constraints.

From Fig. 6, not only do we see confirmation that the PSPS algorithm is able to smooth out the trajectory, but we also see evidence for why that is the case. One reason we expected the robot to be able to improve on kinesthetically taught trajectories was if the teacher was insufficiently coordinated to move the joints simultaneously along the desired trajectories. The regions of roughly constant value in various joints in Fig. 6 suggest the difficulty in manipulating a high-DOF robot kinesthetically. In particular, they show how generally only a few joints can be easily controlled at a time by someone physically moving the robot to teach it a trajectory, and that frictional forces will tend to keep joints not explicitly manipulated at a constant value. PSPS allows the robot to smooth out the times when a joint is active into regions where the joint was not actively manipulated.

We also note that joint LElbowYaw (red line) was hardly moved during kinesthetic training, and that our experiment in PSPS also kept that joint fixed. The fact that PSPS maintained the constancy of that joint is related to the fact that the cost function we used for this experiment was defined *in joint space*. This suggests future work where we experiment with either more complicated cost functions or with cost functions defined in configuration space, as that may enable the robot to improve trajectories by manipulating joints that had been held constant.

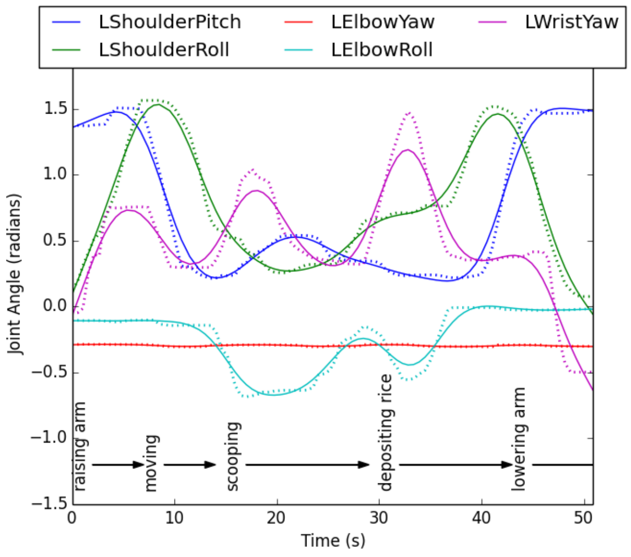

Fig. 6: A comparison of the demonstration joint trajectories (dotted) and the learned joint trajectories (solid lines). Pepper's joints are defined in Fig. 4. General descriptions of motion components are given at the bottom of the chart

VI. Conclusion

In this paper we presented the Parameterized Similar Path Search algorithm, a novel approach to improve kinesthetic trajectory learning for the robotic feeding domain, in which the robot iteratively improves the trajectory demonstrated by the caretaker over a known cost function and requests evaluations from the caretaker to check compliance with unknown task constraints.

We analyzed the performance of PSPS in a two-dimensional simulation as well as on a real Pepper robot performing a rice serving task. We found that the PSPS algorithm is able to successfully improve the training trajectory over the known cost function while satisfying the caretaker's task constraints, even without receiving feedback from the caretaker on the specific moment during trajectory attempts that the trajectory violated task constraints.

The PSPS algorithm, while useful independently, also suggests future avenues of research in which results from several runs of the PSPS algorithm could then be fed into learning-from-demonstration algorithms in order to generalize these more optimal trajectories onto new environments.

References

[1] Aldebaran/SoftBank Robotics. Pepper - documentation. http://doc.aldebaran.com/2-5/home_pepper.html. Accessed: 2018-02-27.

[2] Brenna D Argall, Brett Browning, and Manuela Veloso. Learning robot motion control with demonstration and advice-operators. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 399–404. IEEE, 2008.

[3] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[4] Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20, 1997.

[5] Camanio Care. Bestic. http://www.camanio.com/us/products/bestic/. Accessed: 2018-02-18.

[6] Deborah Christensen et al. Prevalence of cerebral palsy, co-occurring autism spectrum disorders, and motor functioning–autism and developmental disabilities monitoring network, usa, 2008. *Developmental Medicine & Child Neurology*, 56(1):59–65, 2014.

[7] Yan Duan et al. One-shot imitation learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1087–1098. Curran Associates, Inc., 2017.

[8] Peter J Green and Bernard W Silverman. *Nonparametric regression and generalized linear models: a roughness penalty approach*. CRC Press, 1993.

[9] Shervin Javdani, Henny Admoni, Stefania Pellegrinelli, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization for teleoperation and teaming. *arXiv preprint arXiv:1706.00155*, 2017.

[10] Mealtime Partners, Inc. The mealtime partner dining system description. http://www.mealtimepartners.com/dining/mealtime-partner-dining-device.htm. Accessed: 2018-02-18.

[11] Obi. Obi, robotic feeding device designed for home care. https://meetobi.com. Accessed: 2018-02-18.

[12] Daehyung Park, Hokeun Kim, Yuuna Hoshi, Zackory Erickson, Ariel Kapusta, and Charles C Kemp. A multimodal execution monitor with anomaly classification for robot-assisted feeding. In *2016 IEEE International Conference on Robots and Systems (IROS)*, 2017.

[13] Daehyung Park, You Keun Kim, Zackory M. Erickson, and Charles C. Kemp. Towards assistive feeding with a general-purpose mobile manipulator. *CoRR*, abs/1605.07996, 2016.

[14] Chamika Janith Perera, Thilina Dulantha Lalitharatne, and Kazuo Kiguchi. Eeg-controlled meal assistance robot with camera-based automatic mouth position tracking and mouth open detection. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1760–1765. IEEE, 2017.

[15] Performance Health. Meal buddy. https://www.performancehealth.com/meal-buddy-systems. Accessed: 2018-02-18.

[16] Christian H Reinsch. Smoothing by spline functions. *Numerische mathematik*, 10(3):177–183, 1967.

[17] James B Reswick. Development of feedback control prosthetic and orthotic devices. In *Advances in Biomedical Engineering, Volume 2*, pages 139–217. Elsevier, 1972.

[18] Secom Co., Ltd. My spoon meal-assistance robot. https://www.secom.co.jp/english/myspoon/. Accessed: 2018-02-18.

[19] Rui Silva, Miguel Faria, Francisco S. Melo, and Manuela Veloso. Adaptive indirect control through communication in collaborative human-robot interaction. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3617–3622, Sept 2017.

[20] Won-Kyung Song and Jongbae Kim. Novel assistive robot for self-feeding. In *Robotic Systems-Applications, Control and Programming*. InTech, 2012.

[21] Mike Topping. Early experience in the use of the handy 1 robotic aid to eating. *Robotica*, 11(6):525527, 1993.

[22] Mike Topping. An overview of the development of handy 1, a rehabilitation robot to assist the severely disabled. *Journal of Intelligent & Robotic Systems*, 34(3):253–263, 07 2002. Copyright - Kluwer Academic Publishers 2002; Last updated - 2014-08-23.

[23] Gu Ye and Ron Alterovitz. *Demonstration-guided motion planning*, volume 100 of *Springer Tracts in Advanced Robotics*, pages 291–307. Springer Verlag, Germany, 1 2017.

Preprint submitted to 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems. Received March 1, 2018.