Robot motion adaptation through user intervention and reinforcement learning*

Aleksandar Jevtić†, Adrià Colomé, Guillem Alenyà, and Carme Torras

Abstract

Assistant robots are designed to perform specific tasks for the user, but their performance is rarely optimal, hence they are required to adapt to user preferences or new task requirements. In the previous work, the potential of an interactive learning framework based on user intervention and reinforcement learning (RL) was assessed. The framework allowed the user to correct an unfitted segment of the robot trajectory by using hand movements to guide the robot along a corrective path. So far, only the usability of the framework was evaluated through experiments with users. In the current work, the framework is described in detail and its ability to learn from a set of sample trajectories using an RL algorithm is analyzed. To evaluate the learning performance, three versions of the framework are proposed that differ in the method used to obtain the sample trajectories, which are: human-guided learning, autonomous learning, and combined human-guided with autonomous learning. The results show that the combination of the human-guided and autonomous learning achieved the best performance, and although it needed a higher number of sample trajectories than the human-guided learning, it required less user involvement. Autonomous learning alone obtained the lowest reward value and needed the highest number of sample trajectories.

*Pattern Recognition Letters. Accepted for publication: June 16, 2017. doi: 10.1016/j.patrec.2017.06.017

†The authors are with the Institut de Robòtica i Informàtica Industrial, CSIC-UPC, C/ Llorens i Artigas 4-6, Barcelona 08028, Spain. Email: {ajevtic, acolome, galenya, torras} @iri.upc.edu

1 Introduction

Recent studies have shown a growing interest in service robots for various application domains [1]. However, very few robots of this kind can be found in real commercial settings because of their limited ability to adapt to new requirements. Learning from interaction is an attractive strategy for robot learners because it can speed up the learning process and improve its accuracy [2]. Moreover, it allows a user to decide what part of robot's behavior needs to be modified.

Machine learning (ML) is an inseparable component of robotics research. Many ML algorithms were specifically designed for applications in robot vision [3], action planning [4], and motion learning [5]. Initially, a robot's ability to adapt depended on autonomous learning [6], but the advances in human-robot interaction led to development of learning from demonstration algorithms that place the user in the role of a teacher [7].

Learning from demonstration has certain advantages over autonomous robot learning. Most importantly, it allows the user to decide when and how to modify the robot's behavior. In case of robot motion learning, which is the focus of this work, user demonstration can speed up the learning process by reducing the trajectory exploration space. Teaching a robot to optimally perform a task usually requires a lot of data about the task at hand, the environment, or the user preferences on how to perform the task. In such cases, a combination of user demonstration and autonomous learning can produce better results than each of the methods separately.

Robot companions are service robots designed to

Figure 1: Robot assisted-dressing tasks developed under the I-DRESS project. The assistance with putting on a shoe shown on the left is an example of a single-arm robot task, and it is intended for users with reduced mobility. The assistance with gown dressing shown on the right is a dual-arm robot task and represents the case of medical staff whose physical contact with the garments must be limited during dressing to avoid contamination.

help a specific group of users, such as people with disabilities, to live independently [8]. Support to users with the activities of daily living (ADLs) is important but challenging, and the ability to learn from the interaction may improve robot's performance while bringing more comfort to the users [9]. The work presented in this paper was developed under the framework of the I-DRESS project[1] whose main goal is development of a robot assistant for support in dressing (see Fig. 1). Limited sensing and cognitive abilities can significantly reduce robot's performance or even prevent it from executing such task. However, certain aspects of human-robot interaction, such as the robot motion, can be improved with ML.

In the previous work [10], we assessed the usability of an interactive learning framework based on user intervention and reinforcement learning (RL), to allow users to modify an unfitted segment of a robot's trajectory. The motion adjustment was done using hands' gestures to guide the robot along a corrective path. The framework was implemented on a Barrett WAM robot and its performance and user workload were compared in the single-arm and dual-arm setting through a series of experiments with non-expert users. In the current work, besides describing the framework in detail, its ability to learn an improved trajectory from a set of modified sample trajectories is studied. Comparison is made for different levels of user involvement, i.e., autonomous learning, in obtaining the sample trajectories that were used as input for an RL algorithm to produce a new trajectory.

Due to complexity of assisted dressing, at the initial stage the proposed framework was implemented on a simpler robot guidance task, as a proof of concept. This task is shown in Fig. 6, and it is described in detail in the following sections.

1.1 Relevant work

Adaptation of personal robots depends on their ability to naturally interact with users and learn new skills and behaviors from interaction. Communication between the robot and the user is performed through common modalities [11], such as gestures, which are used to recognize user's intentions. Gestures can be used to initiate or stop a robot task, switch between different tasks, but also give insight into the user's behavior. Numerous methods for gesture recognition have been proposed in literature and they were based on statistical modelling, computer vision, pattern recognition, and so forth [12]. Release of affordable RGB-D cameras such as the Microsoft Kinect allowed development of novel algorithms for human body segmentation [13], which improved the performance of the state-of-the-art gesture recognition methods, also for robotics applications [14]. In the current work, a hands-tracking algorithm was applied to robot manipulators teleoperation and was combined with RL to develop a system that is able to learn from user intervention.

Robot learning can be performed at different levels of task hierarchy, such as actions or motions [15]. Some authors suggest that the user teaching and robot learning processes are coupled and proposed scaffolding as a method for user's support in robot learning [16]. It was also shown that the learning performance can be improved by allowing the users to structure the robot task, which allows them to select a task segment they regard as relevant [17].

Learning from a user demonstration can lead to better and more stable robot behaviors, and it can

[1] I-DRESS project: http://www.i-dress-project.eu/

also speed up the learning process. Since the introduction of linearly parameterized trajectories like Dynamic Movement Primitives (DMP) [18], the learning of robot motion has been dominated by Policy Search (PS) algorithms [5], in which the motion policy is directly encoded as the trajectory parametrization. Generally, in PS approaches, the robot is taught an initial trajectory that is then improved through autonomously-generated rollouts and policy updates [7]. A model-based RL proved to be efficient for learning action sequences with the user in the role of a teacher [19]. Instead of applying RL to teach the robot new actions and their effects, in this work, RL is applied to modify the robot trajectory segment selected by the user. In this way, PS can produce a new solution from the user-generated rollout, which is a rather unexploited domain for RL.

In order to apply RL approaches, it is very common to use *ad hoc* evaluations of performance, i.e.: task-oriented reward functions that will provide the relative value of motion executions [7]. The field of Inverse Reinforcement Learning (IRL) [20] finds a proper characterization of a reward function, but most IRL approaches rely on the *perfect teacher* hypothesis, considering human demonstrations as optimal solutions. In the current study, user demonstrations were not considered optimal and the reward function was empirically chosen to minimize the length of the trajectory that avoids the obstacles.

2 Algorithms

The proposed interactive learning framework has two components: interaction and learning. Interaction was implemented through a user intervention algorithm that relies on user's hands-tracking and robot-control algorithms. An RL algorithm is responsible for learning a new robot trajectory from a set of sample trajectories. For the case when the sample trajectories were generated by the user, a time-alignment algorithm was developed to align the trajectories in time, which is a requirement for RL based on policy search. The RL algorithm was implemented in Matlab. The other algorithms were implemented in C++ using the Robot Operating System (ROS) framework.

Three versions of the proposed framework were developed and compared and they differ in the method that was used to generate sample trajectories for the RL algorithm. The first method applies human-guided learning (HGL), in which the sample trajectories were generated by the user, through teleoperation. For each sample trajectory, the user would stop the robot during the execution of the initial trajectory and modify a selected segment using hand gestures. This is represented by the user intervention block in the HGL diagram shown in Fig. 2.

The second method applies the autonomous learning (AL), which represents the case when user intervention is not available (e.g., in case of a user with reduced mobility). AL searches for solutions around the initial trajectory by applying different magnitudes of exploration, which are set empirically by the system designer. The diagram in Fig. 3 shows one policy update of the AL. The AL can be performed for a preset number of policy updates. This number is empirically set and its effect on the learning performance is discussed in Section 4.

Finally, the third method combines human-guided and autonomous learning (HGAL), in which the autonomous learning exploration magnitude is obtained from the user's input. Again, HGL takes several user-generated trajectories to produce a new trajectory, which is then used as input for AL that performs additional trajectory tuning. AL performs a number of policy updates to produce the final new trajectory. The diagram of the HGAL is obtained by connecting the HGL and AL diagrams, in that order, and closing the loop around the AL block, as shown in Fig. 4.

2.1 Robot control

To allow the robot to correctly mimic the human motion, a method based on DMPs was used, similar to the one described in [21]. A diagram of the method is shown in Fig. 5. The forcing term of the DMPs was removed to simplify the shape of the robot trajectory between the directional points. The user's hands-tracking algorithm provided directional points, which were transformed to joints positions that satisfy the kinematics constraints implemented through an in-

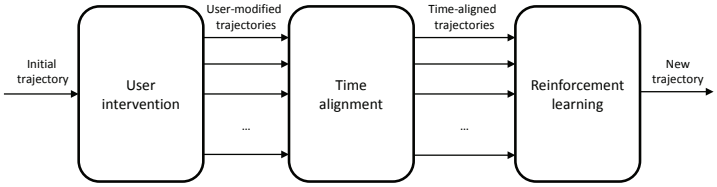


Figure 2: Diagram of the human-guided learning (HGL) method.

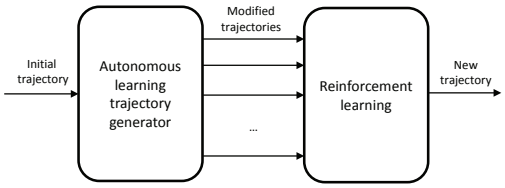


Figure 3: Diagram of the autonomous learning (AL) method.

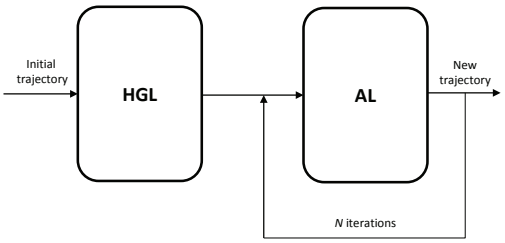


Figure 4: Diagram of the human-guided with autonomous learning (HGAL) method. HGAL combines human-guided learning (HGL) and autonomous learning (AL) by using the new trajectory of HGL as input for AL. The closed loop around the AL block indicates that AL can be executed for an arbitrary number of policy updates (iterations) to improve the final new trajectory.

verse kinematics (IK) algorithm, as in [22]. Joints positions were not provided at the control loop's execution rate, hence a dynamic system was used to interpolate the missing values. In particular, a second-order attractor that works similarly to a DMP [18] tracks the desired joint acceleration $\ddot{\mathbf{p}}$ that is sent as a reference to the robot controller. The reference acceleration is a function of the current position $\mathbf{p}$, velocity $\dot{\mathbf{p}}$, and joint goal $\mathbf{g}$ provided by the IK algorithm, multiplied by proportional-derivative gains:

$$\ddot{\mathbf{p}} = \boldsymbol{K}_p(\mathbf{g} - \mathbf{p}) - \boldsymbol{K}_d\dot{\mathbf{p}}. \quad (1)$$

Such dynamic system characterization permits online trajectory modifications, while keeping all the advantages of the DMPs. The trajectory was recorded as a sequence of Cartesian points obtained from the camera and equally distributed in time, $\mathbf{p}^t = (x^t, y^t, z^t)$, $t = 1 \dots N_t$, making it easy to reproduce the user's hands motions. Cartesian representation also allowed affine transformations of the trajectory, a useful feature for robot motion adaptation in space. Given the joint position reference obtained in (1), a compliant feed-forward controller that combines a friction model with a PID error compensation was used to generate the torque commands, as in [23]. The compliant controller made the robot safer in case of an unexpected physical contact with the user or the surrounding objects. The points $\mathbf{p}^t$ stored in this manner are directional points, meaning that they are not necessarily reached by the robot, but only define robot's motion direction before the next directional point is sent. In practice, these points are provided at a 10Hz rate and the gains used in (1) are tuned to provide a smooth transition between directional points.

In the case of AL, DMPs were initialized by fitting a set of Gaussian weights along the trajectory. Teleoperation by the user consisted of in-real-time modification of the robot trajectory, and equally-spaced

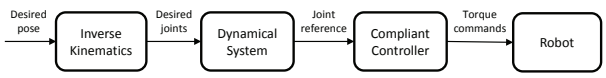


Figure 5: Robot control diagram.

trajectory points were chosen as directional points to represent the robot motion.

2.2 User's hands tracking

User's hands-tracking algorithm was implemented using the OpenNI™and NiTE™open source libraries. The positions of the user's joints were provided in the camera frame of reference. Since relative joints displacements were used, no transformation of the joints positions to the robot frame of reference was needed.

To start modifying a segment of the robot's initial trajectory, the user would stand in front of the robot and raise the right hand to stop the robot and change its state from "EXECUTE" to "FOLLOW". This simple implementation avoided possible deployment issues with, for example, a speech recognition interface. Although speech could allow a more intuitive human-robot interaction, our initial tests showed no considerable impact on the delay when stopping the robot by gesture.

In the "FOLLOW" state, the user could teleoperate the robot by using hand movements. The hands-tracking algorithm recorded the positions of the user's hands that served as directional points for the robot control algorithm described in Section 2.1. The points were reproduced, in real time, by the corresponding robot's end-effector in a two-dimensional horizontal plane above the table, as shown in Fig. 6. The same gesture was used to make the transition back to the "EXECUTE" robot state to stop the teleoperation.

2.3 User intervention

The implementation of the proposed user intervention is given in the Algorithm 1. The user was expected to observe the robot while performing the task, stop it at a desired position by raising the right hand and then, within the next 2 seconds, place the hands in a comfortable posture similar to holding a wheel when driving a car (see Fig. 6).

When switching to the teleoperation mode, i.e., the "FOLLOW" state, the robot would store the initial position of the user's hand $\mathbf{p}_h^0$ and the corresponding end-effector's trajectory cut point, $\mathbf{p}^{I^{cut}}$. The cut point is a directional point from the original trajectory that is closest to the location where the robot was stopped by the user; this point is saved by its index in the trajectory points sequence, I^{cut}. All subsequent displacements of a hand were reproduced by the robot's end-effector towards the new directional points:

$$\mathbf{p}_{new}^t = \mathbf{p}^{I^{cut}} + (\mathbf{p}_h^t - \mathbf{p}_h^0) \tag{2}$$

where $\mathbf{p}_h^t$ is the hand position in moment t recognized by the camera with respect to the initial hand position, $\mathbf{p}_h^0$.

To stop the intervention and relieve control of the robot, the user would again raise the right hand and the current position of the robot end-effector would be stored as the final point of the modified trajectory segment, $\mathbf{p}_{new}^{N_t}$. The robot end-effector would then move to the closest connect point on the original trajectory, $\mathbf{p}^{I^{con}}$, stored by its index I^{con} in the trajectory points sequence, and execute the remaining points of the original sequence until finishing the task.

Original trajectory points were indexed and had associated timestamps to always ensure the identical execution. As previously mentioned, the cut and connect points were stored by their indexes; however, while the cut point was stored as the last trajectory point before the user raised the right hand, the connect point was selected by the minimal distance from the point in which the user relieved control of the robot, as shown in Fig. 7. In the single-arm experiment this calculation was performed for one robot

Algorithm 1 User intervention

```
1: while not end of trajectory τ_0 do
2:     if FOLLOW signal then ▷ user intervention
3:         I_k^cut ← current point index
4:         τ_k ← points 1 ... I_k^cut from τ_0
5:         while not EXECUTE signal do
6:             p^t ← move robot to follow user hands
7:             τ_k ← append p^t
8:         I_k^con ← find closest connect point to τ_0
9:         τ_k ← append points I_k^con ... N_t from τ_0
10:    else
11:        move robot to next τ_0 point
```

Figure 6: A user guiding a dual-arm robot by using hands' gestures tracked by a Microsoft Kinect camera. The original robot trajectory is shown in green (left image). Red arrows show the user's hands' gestures and the corresponding robots' motion in the teleoperation mode (right image).

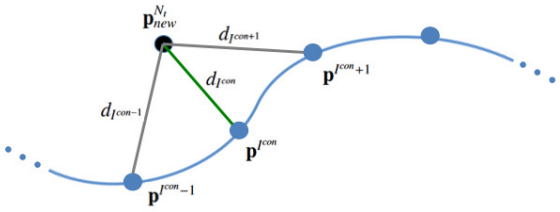


Figure 7: Computation of the connect point between the user trajectory and the original robot trajectory. Blue points belong to the original trajectory, and the final point generated by user teleoperation is shown in black. The curved line represents a general case. In this work, the trajectory used in the experiments was a straight line.

arm only, but in the second experiment involving a dual-arm robot, the connect points were calculated for each robot arm and the index of the one closer to their original trajectories, respectively, was chosen for both robot arms.

The result of the user intervention was a new robot trajectory made of three segments: the unmodified trajectory sequence preceding the cut point, a modified trajectory sequence between the cut and connect points, and the final unmodified trajectory sequence that follows after the connect point. This trajectory, after being aligned in time with other user-modified trajectories, was used as a sample trajectory for the RL algorithm.

2.4 Reinforcement learning

For the trajectories created using the HGL, RL was applied to find an improved trajectory solution by compensating for the perturbations that can result from defective trajectories supplied by the user. The secondary effect was to smooth out the noise in both hands-tracking and robot-control loops. In AL case, due to absence of user intervention, the sample trajectories were generated using an initial variance empirically set by the system designer. The variance was updated after each policy update by a weighted maximum likelihood estimation of a Gaussian distribution. For the combined HGAL, a new trajectory obtained from the user intervention, i.e., the HGL, was used to set the initial variance for the AL exploration.

The reward function defined in (5) was applied to evaluate the performance of the proposed method and the RL policy updates. The directional points trajectory representation chosen in Section 2.1 is very suitable for Policy Search (PS) algorithms [5], where a policy is defined by trajectory parameters (rele-

vant components of the desired poses) and the best policy is obtained by adequately updating these parameters. In particular, the Relative Entropy Policy Search (REPS) was used [24], and for the HGL, only one policy update was performed.

In a group of N modified trajectories $\boldsymbol{\tau}_k = \{\mathbf{p}_k^1, ..., \mathbf{p}_k^{N_t}\}$, $k = 1..N$, each trajectory has its own cut and connect points with associated indexes I_k^{cut}, I_k^{con}, which were obtained as described in Algorithm 1. The REPS algorithm was applied only to the modified segments of the trajectories, between these two points. However, the modified segments were not of the same length since they were not aligned in time. In order to apply RL, these segments were aligned to have the same number of points marked with the same timestamps, as shown in Fig. 8.

First, the minimal cut index, $I_{min}^{cut} = \min_{k=1..N_{rollouts}}(I_k^{cut})$, and the maximal connect index, $I_{max}^{con} = \max_{k=1..N_{rollouts}}(I_k^{con})$, were found among all the sample trajectories. For each trajectory, the segments before I_{min}^{cut} and after I_{max}^{con} were marked as unmodified (shown in black in Fig. 8). Before applying the time alignment and subsequently the RL algorithm, unmodified trajectory segments were removed from the trajectories, leaving only the trajectory segments between the common cut and connect points (shown in blue in Fig. 8), $\mathbf{T}_k = \{\mathbf{p}_k^{I_{min}^{cut}}, \ldots, \mathbf{p}_k^{I_{max}^{con}}\}$, for $k = 1, \ldots, N_{rollouts}$. Points $\mathbf{p}_k^{I_{min}^{cut}}$ and $\mathbf{p}_k^{I_{max}^{con}}$ were included in both modified and unmodified trajectory segments to ensure smooth merging of the segments after applying the time-alignment and RL algorithms.

For the modified trajectories' segments, $\mathbf{T}_k$, a function of time was parametrized with a sum of weighted Gaussians and by using the reward function with the square of the difference from the reference trajectory segment (empirically set to be the first sample trajectory) a gradient descent was performed on the time-deformation function to obtain a better time-aligned trajectory (see Appendix for more details). The time-aligned trajectories are displayed in green color in Fig. 8. The REPS algorithm was used to compute trajectory weights w_k for time-aligned trajectory segments and produce a new trajectory segment (displayed as a thin black curve in Fig. 8),

Algorithm 2 Reinforcement learning of new trajectories

1: Input: HGL or AL sample trajectories $\boldsymbol{\tau}_k = \{\mathbf{p}_k^1, ..., \mathbf{p}_k^{N_t}\}$, for $k = 1 \ldots N_{rollouts}$; cut and connect points indexes I_k^{cut}, I_k^{con}, $\forall k$; reward values of the trajectories R_k, $\forall k$.
2: Find common cut and connect point indexes for all the sample trajectories (rollouts), I_{min}^{cut} and I_{max}^{con}.
3: For all rollouts, extract trajectory segments that will be modified by RL: $\mathbf{T}_k = \{\mathbf{p}_k^{I_{min}^{cut}}, ..., \mathbf{p}_k^{I_{max}^{con}}\}$, $\forall k$.
4: Apply time alignment to $\mathbf{T}_k$, $\forall k$, using the initial rollout as a reference; re-sample to have the same number of points in modified segments from all rollouts.
5: Apply REPS to compute the relative importance weight for each rollout: $w_k = \text{REPS}(\mathbf{R})$, $\forall k$
6: Obtain a new trajectory segment by a weighted sum of the time-aligned, resampled rollouts, $\mathbf{T}_{new}$
7: Merge the unmodified trajectory segments from step 2 with the modified trajectory segment from step 6 to obtain the RL-modified trajectory, $\boldsymbol{\tau}_{new}$

$\mathbf{T}_{new} = \{\mathbf{p}_{new}^{I_{min}^{cut}}, \ldots, \mathbf{p}_{new}^{I_{max}^{con}}\}$, whose points are obtained as follows:

$$\mathbf{p}_{new}^t = \sum_{k=1}^{N_{rollouts}} w_k \mathbf{p}_k^t, \forall t = I_{min}^{cut}, \ldots, I_{max}^{con}. \quad (3)$$

This trajectory segment was then merged with two unmodified segments of the original trajectory that were removed before applying RL, to obtain a complete trajectory:

$$\boldsymbol{\tau}_{new} = \{\mathbf{p}^1, ..., \mathbf{p}^{I_{min}^{cut}-1}, \mathbf{p}_{new}^{I_{min}^{cut}}, ..., \mathbf{p}_{new}^{I_{max}^{con}}, \mathbf{p}^{I_{max}^{con}+1}, ..., \mathbf{p}^{N_t}\}. \quad (4)$$

The data processing steps are summarized in Algorithm 2.

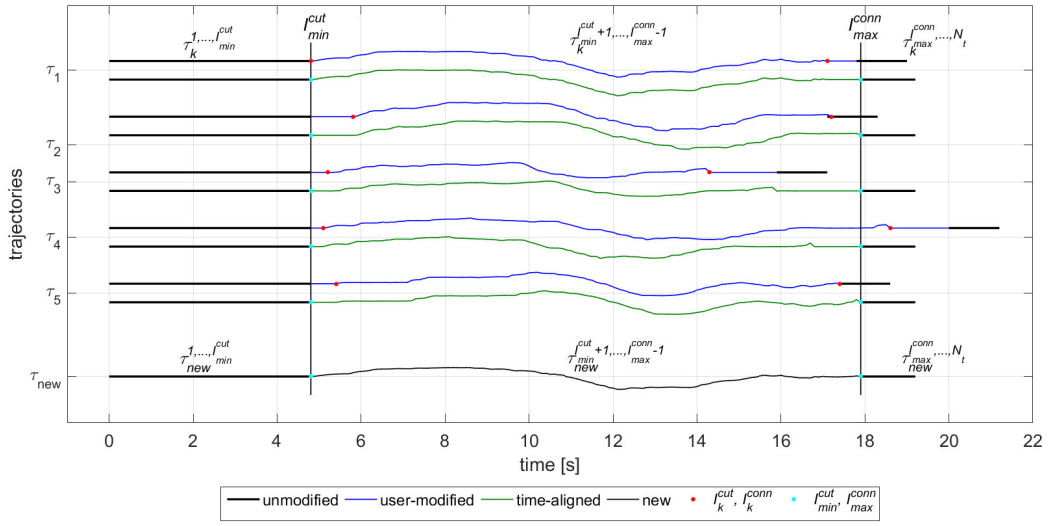


Figure 8: Time alignment of user-modified trajectory segments. Human-guided trajectories (blue) from 5 trials are time-aligned (green) and used as input for the RL algorithm to produce a new trajectory (black).

3 Experimental setup

The proposed algorithms were implemented on two Barrett's 7-DOF Whole Arm Manipulator (WAM) robots and a first-generation Microsoft Kinect camera, which was used for user tracking (see Fig. 6). The experiments included a single-arm and dual-arm robot tasks with a variable level of user intervention. In both tasks, the goal was to modify the robot end-effector trajectory to avoid two obstacles placed on the tabletop. The two robot arms were positioned to face one another from the opposite sides of a square-shaped $1.3m \times 1.3m$ table. The Kinect was mounted on a tripod at the height of $140cm$ facing the user from the opposite side of the table.

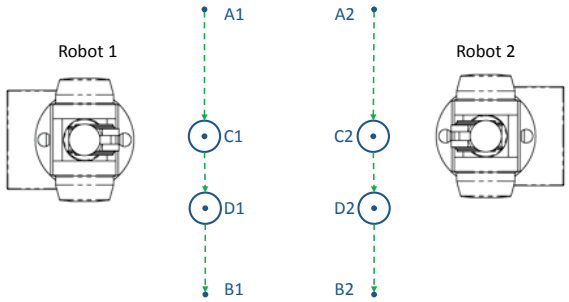


Figure 9: The experimental setup scheme (top view). Obstacles are shown as blue circles and green lines represent the original robots end-effectors trajectories.

For the single-arm task, the robot executed a pre-recorded motion along a straight line (shown in green in Fig. 9), at a constant height from the tabletop, from the point $A1 = (0.5m, 0.4m)$ to the point $B1 = (0.5m, -0.4m)$, in the robot frame of reference. Two obstacles, i.e., plastic bottles shown as blue circles in Fig. 9, were placed along the same line at the point $C1 = (0.5m, 0m)$ and the point $D1 = (0.5m, -0.2m)$. Without modifying the original trajectory, the robot would knock over both bottles, so the goal was to modify a segment of the trajectory to guide the robot around the bottles in an "S"-shaped motion.

The second task, also represented in Fig. 9, in-

volved two robot arms that simultaneously performed motions along two parallel lines. The first robot arm followed the linear trajectory from the point $A1$ to the point $B1$, while the second robot arm performed a parallel motion from the point $A2 = (0.5m, -0.4m)$ to the point $B2 = (0.5m, 0.4m)$, in the first robot and second robot frames of reference, respectively. Four plastic bottles were placed on the tabletop, two along the each of the robot arm's trajectories, at the points $C1$ and $D1$ and the points $C2 = (0.5m, 0m)$ and $D2 = (0.5m, 0.2m)$, in the first robot and second robot frames of reference, respectively. As for the first experiment, the goal was to guide the robot arms around the bottles in a symmetrical "S"-shaped motion. In both experiments, the robot end-effectors had fixed orientation facing down towards the tabletop, and their distance from the tabletop was constant during the whole experiment.

3.1 Performance evaluation

The evaluation of the experimental results was performed using two metrics: task success rate and trajectory reward value.

The task was considered successful if the new trajectory did not knock over any obstacles. The task success rate represents the percentage of the successful trajectories for the same experimental setting (e.g. number of robot arms, policy updates, rollouts per update).

The trajectory reward value obtained with a reward function was also used to assess the performance of our method in each experiment and simulation trial. The reward value penalized the trials that involved longer robot arm trajectories or knocked over the obstacles:

$$R = -\alpha \cdot D_{total} - B \tag{5}$$

where B is the number of knocked-down obstacles during the modified task execution and D_{total} is the total distance travelled by the robot end-effectors. Both B and D_{total} values were normalized (i.e., divided by 2) for the dual-arm robot setting. The factor α is a relative weight chosen to keep the value of B at around 80% of the total reward value; this ratio was suitable to achieve the primary task, i.e., obstacle avoidance, while minimizing the trajectory length. Similar empirical approaches can be found in previous PS applications [5].

4 Results and Discussion

The experiments with all three versions of the proposed interactive learning framework were performed, and their results are here compared using the metrics defined in Section 3.1.

4.1 Human-guided learning (HGL)

In this experiment, a user with no experience in robotics performed 20 trials of modifying the original robot arms trajectories for both single-arm and dual-arm tasks, as described in Algorithm 1. Before the experiment, the user was allowed to test the basic robot features. The new trajectories were obtained using Algorithm 2 with only one policy update after 5, 10, 15, and 20 trials.

Both single-arm and dual-arm tasks were performed with a 100% task success rate, with the exception of the single-arm experiment when Algorithm 2 was applied after only 5 trials.

The performance of the new trajectories was further evaluated using the reward function defined in (5) and the obtained values are shown in Table 1.

Both experiments yielded high reward values, with the exception of the single-arm experiment when Algorithm 2 was applied after only 5 trials ($R = -1.127$). This was a consequence of lower reward values in the initial trials in which the user was becoming familiar with the robot. Still, even this trajectory was a reasonably good solution because it only slightly touched one of the bottles. Importantly, for

Table 1: Reward values obtained by HGL for different number of trials

Task	$N = 5$	$N = 10$	$N = 15$	$N = 20$
Single-arm	-1.127	-0.116	-0.103	-0.099
Dual-arm	-0.635	-0.109	-0.105	-0.103

Table 2: Task success rate over 100 simulations for AL after different number of trials (rollouts per policy update)

Task	$N = 5$	$N = 10$	$N = 15$	$N = 20$	$N = 50$	$N = 100$	$N = 200$	$N = 500$	$N = 1000$
Single-arm	24%	34%	42%	75%	90%	98%	100%	100%	100%
Dual-arm	4%	12%	12%	24%	30%	42%	64%	76%	100%

both tasks, the increasing trend of the reward values after each policy update shows a continuous learning process toward a better solution.

4.2 Autonomous learning (AL)

AL method was tested in a simulated environment, using the same setup as for the HGL, and the trajectory representation described in Section 2.1. In the initial tests, white and red noise were added to the original trajectory points to generate modified trajectories. However, this approach failed and the RL algorithm was unable to produce a trajectory that would circle around the obstacles without knocking them over.

For this reason, DMP was applied as more suitable for random generation of trajectories. Note that the DMP characterization of the whole trajectory was not used in case of the HGL because modification of trajectories was performed during task execution; DMP parameters are usually obtained by means of least squares minimization applied to the whole trajectory, which prevented re-parametrization in real time. In each simulation, 100 REPS policy updates were performed with different numbers of rollouts per update.

Table 2 shows the success rate of the single-arm and dual-arm tasks over 100 simulations. For example, 24% rate for the single-arm task, in case of 5 rollouts per policy update, means that 24 out of 100 simulations produced trajectories that were successful and did not knock over the obstacles. A trend of improving the task success rate can be noted when the number of rollouts was increased. Interestingly, the dual-arm task proved to be complex and required 1000 rollouts per policy update to obtain a 100% success rate. The generation of random trajectories for the two arms is seen as independent events; therefore, the probability of not knocking down an obstacle with two arms is the product of probabilities of not knocking down an obstacle with each arm. For the tasks that require symmetrical motion of the arms, the learning process could be simplified to single-arm learning (while the other arm performs a mirror motion) to improve the results. However, this would affect the generalization of such results.

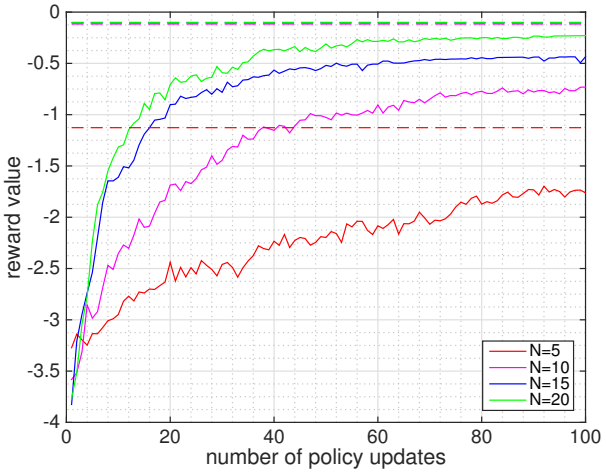
The simulation results after applying AL to single-arm and dual-arm tasks are shown in Fig. 10a and 10b, respectively. Learning curves represent the average reward values obtained over 100 simulations, for different numbers of policy updates and rollouts per update. Dashed lines represent the reward values obtained after applying HGL (see Table 1). The results show that even one policy update for HGL yields higher reward values than what was obtained as the average of 100 AL simulations, except for the initial 5 trials of the single-arm task when the user was unfamiliar with the robot.

Table 3: Task success rate over 100 simulations for HGAL (N is the number of rollouts used for HGL while AL was performed with 100 policy updates and 20 rollouts per update)

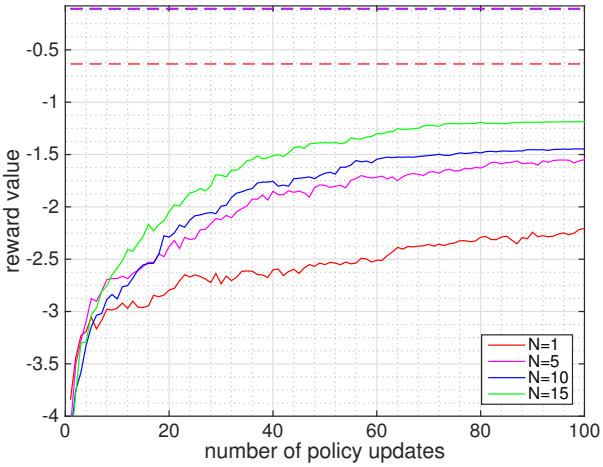
Task	$N = 1$	$N = 5$	$N = 10$	$N = 15$	$N = 20$
1-arm	98.8%	100%	100%	100%	100%
2-arm	49.9%	84%	90%	92%	96%

4.3 Human-guided with autonomous learning (HGAL)

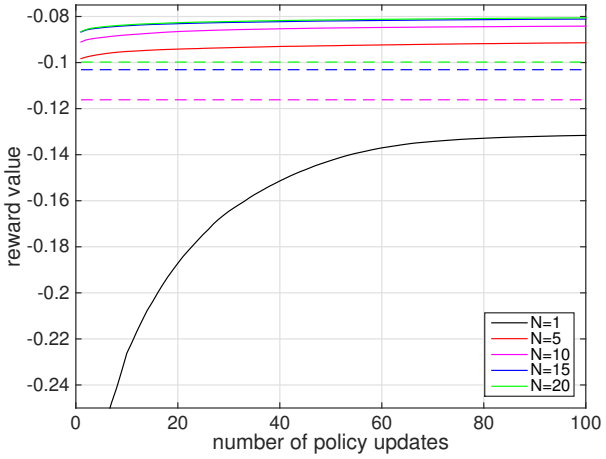
In the combined approach of HGL and AL, the output trajectory of the HGL was used as the initial trajectory for the AL. Table 3 shows the average performance of the single-arm and dual-arm tasks over 100 simulations. Note that the number of rollouts



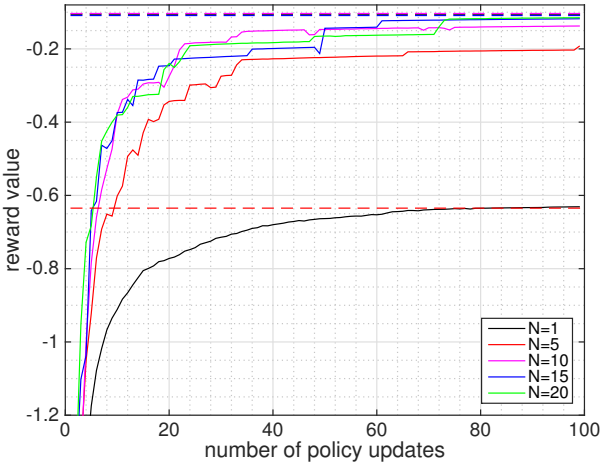
(a) Single-arm task, AL vs. HGL



(b) Dual-arm task, AL vs. HGL



(c) Single-arm task, HGAL vs. HGL



(d) Dual-arm task, HGAL vs. HGL

Figure 10: Comparison of the learning curves showing reward values obtained from AL and HGAL executions after different numbers of policy updates, and rollouts per update. Horizontal lines represent reward values obtained from the HGL experiments given in Table 1. Note that some of these lines overlap, and that in the figure (c) the value $R = -1.127$ obtained after a policy update with 5 rollouts is not shown due to space limitation. Figures (c) and (d) also show (in black) the average learning curve after applying HGAL when only one human-guided rollout was used to generate the initial trajectory for the autonomous learning part.

represents the total number of HGL rollouts used to generate the initial trajectory for AL, but AL was performed with the 20 rollouts per policy update. This produced a sufficiently high task performance rate; for example, in case of the dual-arm task, after using the trajectory obtained using HGL with 1 policy update and 20 rollouts per update as input, the task performance of the AL applying 100 policy updates and 20 rollouts per update increased from 24% to 96%.

Comparison of the reward values obtained after applying HGAL and HGL is shown in Fig. 10c and 10d. The learning curves show average reward values over 100 simulations with different numbers of policy updates and rollouts per update. It can be noted that, in case of the single-arm task, the HGAL easily outperforms the HGL, or in other words, improves the solution of the HGL. However, for the dual-arm task this was not the case, and although some trajectories obtained by the HGAL were better than the solution of the HGL on average the HGAL yielded slightly lower reward values than the HGL. On the same figures, displayed as a black solid curve, the average reward values are shown that were obtained after using the HGL with 1 rollout per update; however, these results were unsatisfactory because of the limited learning potential of only one rollout.

Finally, the best trajectories for the single-arm and dual-arm tasks obtained after applying all three approaches are shown in Fig. 11. As mentioned earlier, the best results were obtained after 1 policy search update with 20 rollouts (HGL), 100 policy search updates with 1000 simulated rollouts (AL), and 100 policy search updates with 20 simulated rollouts (HGAL). It is important to note that, for the REPS algorithm, better solutions can be obtained by increasing the number of rollouts and policy updates. The quality of trajectories was evaluated using Eq. (5), i.e., the best trajectories are those that avoid the obstacles while performing the shortest path, regardless of their smoothness (see Fig. 11).

HGAL combines HGL and AL in a way that an approximation of the HGL result is used as input for AL. This strategy substantially increased the task success rate, as well as the average rewards, as shown in Fig. 10. Moreover, Fig. 10c and 10d show how the HGAL approach is capable of strongly improving some trajectories, especially in case of unsuccessful ($N = 1$) or poor-performing ($N = 5$) HGL trials.

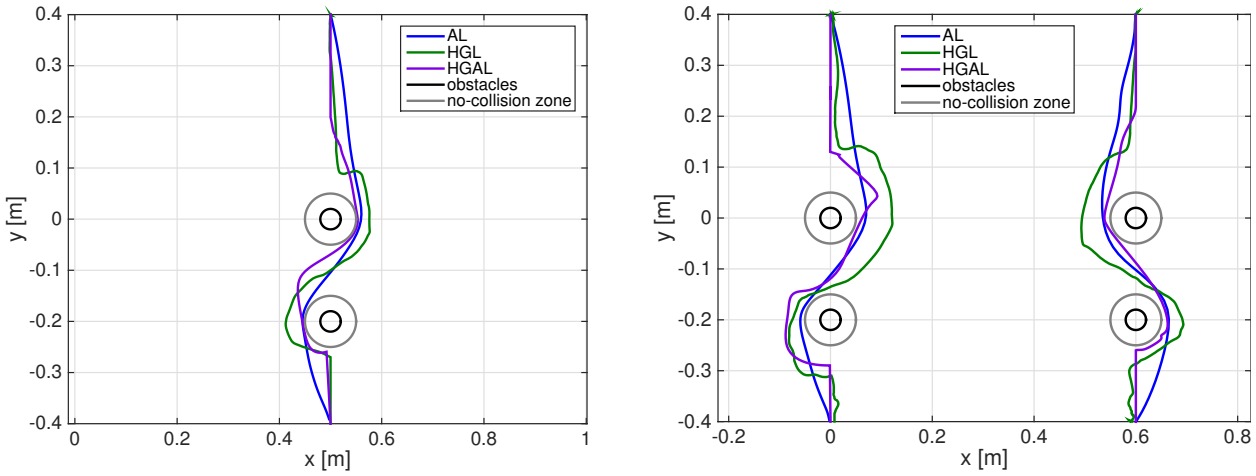
5 Conclusions

Learning through interaction is a must-have ability of personal robot assistants as they must be able to adapt to their users' needs. In this work, an interactive learning framework was proposed that combines user intervention and reinforcement learning, and allows such adaptation. The framework was compared using three learning methods that differ in the level of user intervention, i.e., autonomous learning. The objective was to find the most efficient method to improve the robot motion trajectory and yield the best task performance. The framework was implemented on single-arm and dual-arm robots in the experiment with an untrained user.

The results show that human-guided learning could yield a better solution after using only 1% of the total number of policy updates required for autonomous learning (HGL using 1 policy update vs. AL using 100 policy updates). In some cases, a combined approach of human-guided and autonomous learning could lead to better solutions, but this depended on the quality of the initial solution provided by the user.

The current work is a proof of concept. Importantly, the results showed that the autonomous learning relies on a larger size of input data, i.e., sample trajectories, and that the user input can speed up the learning process. Whether the human-guided learning can be improved with autonomous learning will depend on the robot task complexity and the quality of the initial user input. Nevertheless, the proposed framework is suitable for life-long learning as it allows additional inputs from the user, which has proved to yield better solutions.

As a part of the future work, the proposed framework will be further developed for assistance in activities of daily living, such as dressing. For customization of such tasks to each individual user requirements, an input from the user will be of great value.



(a) Single-arm experiment

(b) Dual-arm experiment

Figure 11: Comparison of the best trajectories for human-guided learning (HGL), autonomous learning (AL) and combined human-guided and autonomous learning (HGAL). The no-collision zone adds the robot arm radius around obstacles.

Acknowledgments

This work was funded by the EU CHIST-ERA I-DRESS project, reference num. PCIN-2015-147, and partially supported by the Spanish national project RobInstruct, reference num. TIN2014-58178-R.

References

[1] C. Torras, "Service robots for citizens of the future," *European Review*, vol. 24, pp. 17–30, 002 2016.

[2] M. Cakmak, C. Chao, and A. L. Thomaz, "Designing interactions for robot active learners," *IEEE Transactions on Autonomous Mental Development*, vol. 2, pp. 108–118, June 2010.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[4] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[5] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Found. Trends Robot*, vol. 2, pp. 1–142, Aug. 2013.

[6] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[7] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, pp. 469–483, May 2009.

[8] T. J. Prescott, T. Epton, V. Evers, K. McKee, W. Hawley, M., T., and P. Dario, "Robot companions for citizens: roadmapping the potential for future robots in empowering older people," in *Bridging Research in Ageing and ICT Development (BRAID)*, may 2012.

[9] G. Canal, G. Alenyà, and C. Torras, *Personalization Framework for Adaptive Robotic Feeding*

Assistance, pp. 22–31. Cham: Springer International Publishing, 2016.

[10] A. Jevtić, A. Colomé, G. Alenyà, and C. Torras, *User Evaluation of an Interactive Learning Framework for Single-Arm and Dual-Arm Robots*, pp. 52–61. Cham: Springer International Publishing, 2016.

[11] M. N. Nicolescu and M. J. Mataric, "Learning and interacting in human-robot domains," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 31, pp. 419–430, Sep 2001.

[12] S. Mitra and T. Acharya, "Gesture recognition: A survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, pp. 311–324, May 2007.

[13] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Commun. ACM*, vol. 56, pp. 116–124, Jan. 2013.

[14] A. Jevtić, G. Doisy, Y. Parmet, and Y. Edan, "Comparison of interaction modalities for mobile indoor robot guidance: Direct physical interaction, person following, and pointing control," *IEEE Transactions on Human-Machine Systems*, vol. 45, pp. 653–663, Dec 2015.

[15] J. Saunders, D. S. Syrdal, K. L. Koay, N. Burke, and K. Dautenhahn, "Teach me – show me: End-user personalization of a smart home and companion robot," *IEEE Transactions on Human-Machine Systems*, vol. 46, pp. 27–40, Feb 2016.

[16] A. L. Thomaz and C. Breazeal, "Teachable robots: Understanding human teaching behavior to build more effective robot learners," *Artif. Intell.*, vol. 172, pp. 716–737, Apr. 2008.

[17] N. Otero, J. Saunders, K. Dautenhahn, and C. L. Nehaniv, "Teaching robot companions: The role of scaffolding and event structuring," *Connect. Sci*, vol. 20, pp. 111–134, June 2008.

[18] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 2, pp. 1398–1403, 2002.

[19] D. Martínez, G. Alenyà, and C. Torras, "Relational reinforcement learning with guided demonstrations," *Artificial Intelligence*, pp. –, 2015.

[20] S. Zhifei and E. Meng Joo, "A survey of inverse reinforcement learning techniques," *International Journal of Intelligent Computing and Cybernetics*, vol. 5, no. 3, pp. 293–311, 2012.

[21] F. Husain, A. Colomé, B. Dellen, G. Alenyà, and C. Torras, "Realtime tracking and grasping of a moving object from range video," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2617–2622, May 2014.

[22] A. Colomé and C. Torras, "Closed-loop inverse kinematics for redundant robots: Comparative assessment and two enhancements," *IEEE/ASME Transactions on Mechatronics*, vol. 20, pp. 944–955, April 2015.

[23] A. Colomé, A. Planells, and C. Torras, "A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5649–5654, May 2015.

[24] J. Peters, K. Mülling, and Y. Altün, "Relative entropy policy search," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)* (M. Fox and D. Poole, eds.), pp. 1607–1612, AAAI Press, 2010.

[25] S. Ruder, "An overview of gradient descent optimization algorithms," *Computing Research Repository (CoRR)*, vol. abs/1609.04747, 2016.

A Time-alignment algorithm

The steps of the time-alignment algorithm applied in Section 2.4 are here described:

1. The algorithm input is a set of $N_{rollots}$ trajectories with different number of points, N_k, in each trajectory, $\mathbf{T}_k = \{\mathbf{p}_k^1, ..., \mathbf{p}_k^{N_k}\}$. Using the first trajectory in the set, $\mathbf{T}_1$, as a reference, the minimum and maximum values of each coordinate j and for each trajectory k are obtained: $\mathbf{p}^k_{\min}(j), \mathbf{p}^k_{\max}(j)$, $\forall j$, $\forall k$.

2. Each trajectory $\mathbf{T}_k$, $k = 1..N_{rollouts}$ is normalized with respect to the reference trajectory, $\mathbf{T}_1$, to obtain a normalized trajectory, $\hat{\mathbf{T}}_k = \{\hat{\mathbf{p}}_1^k, ..., \hat{\mathbf{p}}_{N_k}^k\}$, where $\hat{\mathbf{p}}_t^k$ is

$$\begin{aligned} \hat{\mathbf{p}}_t^k(j) &= \left[\mathbf{p}_t^k(j) - \mathbf{p}^k_{\min}(j)\right] \cdot \\ &\cdot \frac{\mathbf{p}^1_{\max}(j) - \mathbf{p}^1_{\min}(j)}{\mathbf{p}^k_{\max}(j) - \mathbf{p}^k_{\min}(j)} + \mathbf{p}^k_{\min}(j) \end{aligned} \tag{6}$$

   The reason for normalization is putting trajectories with relative offsets into a common scale, this way giving priority to the shape alignment instead of the magnitude alignment.

3. Once rescaled trajectories, $\mathbf{T}_1, \hat{\mathbf{T}}_2, ..., \hat{\mathbf{T}}_{N_{rollouts}}$, are computed, a cubic spline $\mathbf{S}_k$ is fitted through all the points of each trajectory k. This allows the normalization of the time scale to $[0, 1]$ and for trajectories to be evaluated at any point in time.

4. Once all trajectories have been normalized with respect to the reference trajectory, $\mathbf{T}_1$, and encapsulated as splines, each trajectory , $\hat{\mathbf{T}}_k$, $k = 2..N_{rollouts}$ is aligned with the reference one, independently. To do so, the reference spline, $\mathbf{S}_1$, is matched with $\mathbf{S}_k$ by matching $\mathbf{S}_1(t)$ and $\mathbf{S}_k(h(t))$, for several time values, where $h(t, \boldsymbol{\omega})$ is a time transformation parametrized with $\boldsymbol{\omega}$, whose derivative is a weighted sum of Gaussians equally-spaced in time, $\phi(t)$:

$$h'(t) = \sum_{s=1}^{10} \omega_s \phi(t), \tag{7}$$

   where the weights $\boldsymbol{\omega} = [w_1, ...w_{10}]$, $s = 1..10$ are initialized so that $h'(t, \boldsymbol{\omega}_0) \sim 1$, $\forall t$.

5. Gradient descent is performed on the weights using the following cost function C:

$$\begin{aligned} C &= \textstyle\sum_{i_t=0}^{i_t=Nt} \|\mathbf{S}_1(dt \cdot i_t) - \mathbf{S}_k(dt \cdot i_t)\|^2 + \\ &+ 10 \cdot \mathbf{1}_{h(dti_t) < h(dt(i_t-1))} \cdot \\ &\cdot \operatorname{abs}\left(h(dti_t) - h(dt(i_t - 1))\right), \end{aligned} \tag{8}$$

   where $\mathbf{1}$ is the indicator function. The cost function minimizes the vertical difference between both trajectories, but also ensures that the resulting time transformation h is monotonically growing. The gradient descent is performed such that:

$$\boldsymbol{\omega}^{\text{new}} = \boldsymbol{\omega} + \alpha \nabla_{\mathbf{w}} C, \tag{9}$$

   where α is initialized with a small value of 0.1 and increased by a factor of 1.05 if the cost is reduced, while decreased by a factor of 0.8 if the cost is increased. This ensures that the gradient descent algorithm do not oscillate around a solution, hence it converges faster [25].

6. The resulting weights, $\boldsymbol{\omega}_k$, computed from (9) are obtained for each trajectory $k = 2..N_{rollouts}$. The weights are used to evaluate the splines calculated in step 4, $\mathbf{S}_k(h(t; \boldsymbol{\omega}_k))$, at the time steps corresponding to the reference trajectory. The final time-aligned trajectories are displayed in green in Fig. 8.