frontiers in Robotics and AI

ORIGINAL RESEARCH
published: 05 October 2017
doi: 10.3389/frobt.2017.00045

Prediction of Intention during Interaction with iCub with Probabilistic Movement Primitives

Oriane Dermy[1,2,3]*, *Alexandros Paraschos*[4,5], *Marco Ewerton*[4], *Jan Peters*[4,6], *François Charpillet*[1,2,3] and *Serena Ivaldi*[1,2,3]

[1] Inria, Villers-lés-Nancy, France, [2] Université de Lorraine, Loria, UMR7503, Vandoeuvre, France, [3] CNRS, Loria, UMR7503, Vandoeuvre, France, [4] TU Darmstadt, Darmstadt, Germany, [5] Data Lab, Volkswagen Group, Munich, Germany, [6] Max Planck Institute for Intelligent Systems, Tubingen, Germany

This article describes our open-source software for predicting the intention of a user physically interacting with the humanoid robot iCub. Our goal is to allow the robot to infer the intention of the human partner during collaboration, by predicting the future intended trajectory: this capability is critical to design anticipatory behaviors that are crucial in human–robot collaborative scenarios, such as in co-manipulation, cooperative assembly, or transportation. We propose an approach to endow the iCub with basic capabilities of intention recognition, based on Probabilistic Movement Primitives (ProMPs), a versatile method for representing, generalizing, and reproducing complex motor skills. The robot learns a set of motion primitives from several demonstrations, provided by the human via physical interaction. During training, we model the collaborative scenario using human demonstrations. During the reproduction of the collaborative task, we use the acquired knowledge to recognize the intention of the human partner. Using a few early observations of the state of the robot, we can not only infer the intention of the partner but also complete the movement, even if the user breaks the physical interaction with the robot. We evaluate our approach in simulation and on the real iCub. In simulation, the iCub is driven by the user using the Geomagic Touch haptic device. In the real robot experiment, we directly interact with the iCub by grabbing and manually guiding the robot's arm. We realize two experiments on the real robot: one with simple reaching trajectories, and one inspired by collaborative object sorting. The software implementing our approach is open source and available on the GitHub platform. In addition, we provide tutorials and videos.

Keywords: robot, prediction, intention, interaction, probabilistic models

OPEN ACCESS

Edited by:
Lorenzo Jamone, Queen Mary University of London, United Kingdom

Reviewed by:
Rodolphe Gelin, Aldebaran Robotics, France
Francesco Rea, Fondazione Istituto Italiano di Tecnologia, Italy
Mirko Wächter, Karlsruhe Institute of Technology, Germany

Correspondence:
Oriane Dermy
oriane.dermy@inria.fr

Specialty section:
This article was submitted to Humanoid Robotics, a section of the journal Frontiers in Robotics and AI

Received: *31 May 2017*
Accepted: *21 August 2017*
Published: *05 October 2017*

Citation:
Dermy O, Paraschos A, Ewerton M, Peters J, Charpillet F and Ivaldi S (2017) Prediction of Intention during Interaction with iCub with Probabilistic Movement Primitives. Front. Robot. AI 4:45. doi: 10.3389/frobt.2017.00045

1. INTRODUCTION

A critical ability for robots to collaborate with humans is to predict the intention of the partner. For example, a robot could help a human fold sheets, move furniture in a room, lift heavy objects, or place wind shields on a car frame. In all these cases, the human could begin the collaborative movement by guiding the robot, or by leading the movement in the case that both human and robot hold the object. It would be beneficial for the performance of the task if the robot could infer the intention of the human as soon as possible and collaborate to complete the task without requiring any further assistance. This scenario is particularly relevant for manufacturing (Dumora et al., 2013), where robots could help human partners in carrying a heavy or unwieldy object, while humans could guide the robot without effort in executing the correct trajectory for positioning the object at the right location.[1] For example, the human could start moving the robot's end effector toward the goal location and release the grasp on the robot when the robot shows that it is capable of reaching the desired goal location without human intervention. Service and manufacturing scenarios offer a wide set of examples where collaborative actions can be initiated by the human and finished by the robot: assembling objects parts, sorting items in the correct bins or trays, welding, moving objects together, etc. In all these cases, the robot should be able to predict the goal of each action and the trajectory that the human partner wants to do for each action. To make this prediction, the robot should use all available information coming from sensor readings, past experiences (prior), human imitation and previous teaching sessions, or collaborations. Understanding and modeling the human behavior, exploiting all the available information, is the key to tackle this problem (Sato et al., 1994).

To predict the human intention, the robot must identify the current task, predict the user's goal, and predict the trajectory to achieve this goal. In the human–robot interaction literature, many keywords are associated with this prediction ability: inference, goal estimation, legibility, intention recognition, and anticipation.

Anticipation is the ability of the robot to choose the right thing to do in a current situation (Hoffman, 2010). To achieve this goal, the robot must predict the effect of their action, as studied with the concept of affordances (Sahin et al., 2007; Ivaldi et al., 2014b; Jamone et al., 2017). It also must predict the human intention, which means estimating the partner's goal (Wang et al., 2013; Thill and Ziemke, 2017). Finally, it must be able to predict the future events or states, e.g., being able to simulate the evolution of the coupled human–robot system, as it is frequently done in model predictive control (Ivaldi et al., 2010; Zube et al., 2016) or in human-aware planning (Alami et al., 2006; Shah et al., 2011).

It has been posited that having legible motions (Dragan and Srinivasa, 2013; Busch et al., 2017) helps the interacting partners in increasing the mutual estimation of the partner's intention, increasing the efficiency of the collaboration.

Anticipation requires thus the ability to visualize or predict the future desired state, e.g., where the human intends to go to. Predicting the user intention is often formulated as predicting the target of the human action, meaning that the robot must be able to predict at least the goal of the human when the two partners engage in a joint reaching action. To make such prediction, a common approach is to consider each movement as an instance of a particular skill or goal-directed movement primitive.

In the past decade, several frameworks have been proposed to represent movements primitives, frequently called *skills*, the most notable being Gaussian Mixture Models (GMM) (Khansari-Zadeh and Billard, 2011; Calinon et al., 2014), Dynamic Movement Primitives (DMP) (Ijspeert et al., 2013), Probabilistic Dynamic Movement Primitive (PDMP) (Meier and Schaal, 2016), and Probabilistic Movement Primitives (ProMP) (Paraschos et al., 2013a). For a thorough review of the literature, we refer the interested reader to Peters et al. (2016). Skill learning techniques have been applied to several learning scenarios, such as playing table tennis, writing digits, and avoiding obstacles during pick and place motions. In all these scenarios, the humans are classically providing the demonstrations (i.e., realizations of the task trajectories) by either manually driving the robot or through teleoperation, following the classical paradigm of imitation learning. Some of them have been also applied to the iCub humanoid robot: for example, Stulp et al. (2013) used DMPs to adapt a reaching motion online to the variable obstacles encountered by the robot arm, while Paraschos et al. (2015) used ProMPs to learn how to tilt a grate including torque information.

Among the aforementioned techniques, ProMPs stand out as one of the most promising techniques for realizing intention recognition and anticipatory movements for human–robot collaboration. They have the advantage, with respect to the other methods, of capturing by design the variability of the human demonstrations. They also have useful structural properties, as described by Paraschos et al. (2013a), such as co-activation, coupling, and temporal scaling. ProMPs have already been used in human–robot coordination for generating appropriate robot trajectories in response to initiated human trajectories (Maeda et al., 2016). Differently from DMPs, ProMPs do not need the information about the final goal of the trajectory, which is something that DMPs use to set an attractor that guarantees convergence to the final goal.[2] Also, they perform better in presence of noisy measurements or sparse measurements, as discussed in Maeda et al. (2014).[3] In a recent paper, Meier and Schaal (2016) proposed a method called PDMP (Probabilistic Dynamic Movement Primitive). This method improves DMP with probabilistic properties to measure the likelihood that the movement primitive is executed correctly and to perform inference on sensor measurement. However, The PDMPs do not have a data-driven generalization and can deviate arbitrarily from the demonstrations. These last differences can be critical for our humanoid robot (for example, if it collides with something during the movement, or if during the movement it holds something that can fall down due to a bad trajectory, etc.). Thus, the ProMPs method is more suitable for our applications.

In this article, we present our approach to the problem of predicting the intention during human–robot physical interaction and collaboration, based on Probabilistic Movement Primitives (ProMPs) (Paraschos et al., 2013a), and we present the associated open-source software code that implements the method for the iCub.

To illustrate the technique, the exemplifying problem we tackle in this article is to allow the robot to finish a movement initiated by the user that physically guides the robot arm. From the first observations of the joint movement, supposedly belonging to a

[1] Currently, this scenario is frequently addressed in manufacturing by robots and lifters; in the future, we imagine that humanoid robots could also be used for such task, for assisting workers in environments where robots cannot be installed on a fixed base, such as in some aircraft manufacturing operations (Caron and Kheddar, 2016).

[2] There may be applications where converging to a unique and precise goal could be a desirable property of the robot's movement. However, it is an assumption that prevents us to generalize the method for different actions, and this is another reason why we prefer ProMPs.

[3] We refer the interested reader to Maeda et al. (2014) for a thorough comparison between DMPs and ProMPs to be used for interaction primitives and prediction.

movement primitive of some task, the robot must recognize which kind of task the human is doing, predict the "future" trajectory, and complete the movement autonomously when the human releases the grasp on the robot.[4]

To achieve this goal, the robot first learns the movement primitives associated with the different actions/tasks. We choose to describe these primitives with ProMPs, as they are able to capture the distribution of demonstrations in a probabilistic model, rather than with a unique "average" trajectory. During interaction, the human starts physically driving the robot to perform the desired task. At the same time, the robot collects observations of the task. It then uses the prior information from the ProMP to compute a prediction of the desired goal together with the "future" trajectory that allows it to reach the goal.

A conceptual representation of the problem is shown in **Figure 1**. In the upper part of this figure, we represent the training step for one movement primitive: the robot is guided by the human partner to perform a certain task, and several entire demonstrations of the movement that realizes the task are collected. Both kinematics (e.g., Cartesian positions) and dynamics (e.g., wrenches) information are collected. The N trajectories constitute the base for learning the primitive, that is learning the parameters $\boldsymbol{\omega}$ of the trajectory distribution. We call this learned distribution the *prior distribution*. If multiple tasks are to be considered, then the process is replicated such that we have one ProMP for every task. The bottom of the figure represents the inference step. From the *early observations*[5] of a movement initiated by the human partner, the robot first recognizes which ProMP best matches the early observations (i.e., it recognizes the primitives that the human is executing, among the set of known primitives). Then, it estimates the future trajectory, given the early observations (e.g., first portion of a movement) and the prior distribution, computing the parameters $\boldsymbol{\omega}^*$ of the *posterior distribution*. The corresponding trajectory can be used by the robot to autonomously finish the movement, without relying on the human.

In this article, we describe both the theoretical framework and the software that is used to perform this prediction. The software is currently implemented in Matlab and C++; it is open source, available on `github`:

https://github.com/inria-larsen/icubLearningTrajectories

and it has been tested both with a simulated iCub in Gazebo and the real iCub. In simulation, physical guidance is provided by the Geomagic Touch[6]; on the real robot, the human operator simply grabs the robot's forearm.

We also provide a practical example of the software that realizes the exemplifying problems. In the example, the recorded trajectory is composed of both the Cartesian position and the forces at the end effector. Notably, in previous studies (Paraschos et al., 2015), ProMPs were used to learn movement primitives using joint positions. Here, we use Cartesian positions instead of joints positions to exploit the redundancy of the robotic arm in performing the desired task in the 3D space. At the control level of the iCub, this choice requires the iCub to control its lower-level (joint torque) movement with the Cartesian controller (Pattacini et al., 2010) instead of using the direct control at joint level. As for the forces, we rely on a model-based dynamics estimation that exploits the 6 axis force/torque sensors (Ivaldi et al., 2011; Fumagalli et al., 2012). All details for the experiments are presented in the article and the software tutorial.

[4] To avoid disambiguation, in our method, tasks are encoded by primitives that are made of trajectories: this is a very classical approach for robot learning techniques and in general techniques based on primitives. Of course this is a simplification, but it allows representing a number of different tasks: pointing, reaching, grasping, gazing, etc.

[5] In this article, we denote by *early observations* the first portion of a movement observed by the robot, i.e., from $t = 0$ to acurrent t.

[6] The Geomagic Touch is a haptic device, capable of providing force feedback from the simulation to the operator. In our experiments with the simulated iCub, we did not use this feature. We used the Geomagic Touch to steer the arm of the simulated robot. In that sense, we used it more as a joystick for moving the left arm.

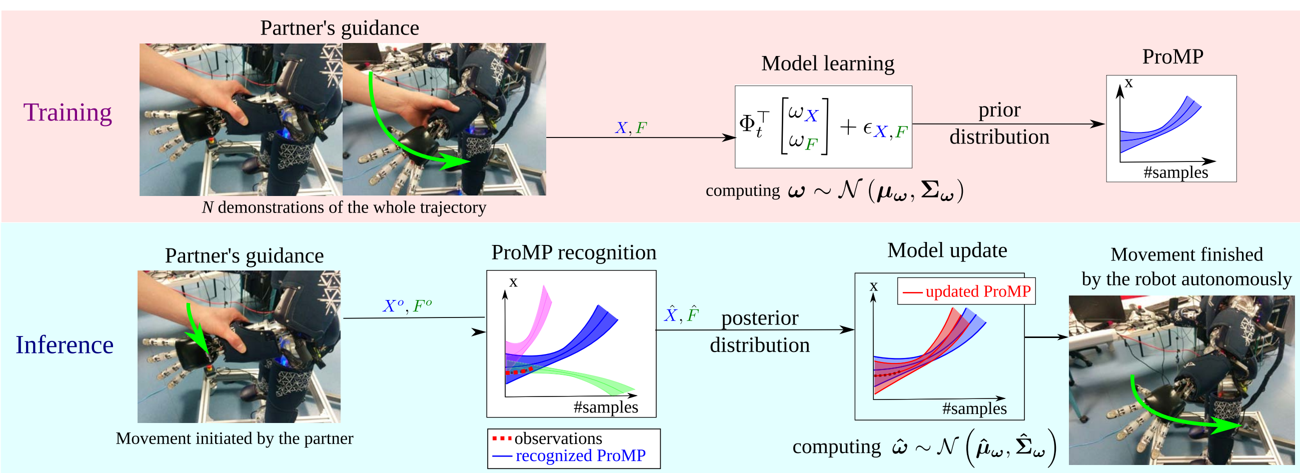

FIGURE 1 | Conceptual use of the ProMP for predicting the desired trajectory to be performed by the robot in a collaborative task. Top: training phase, where ProMPs are learned from several human demonstrations. Bottom: inference phase (online), where from early observations, the robot recognizes the current (among the known) ProMP and predicts the human intention, i.e., the future evolution of the initiated trajectory.

To summarize, the contributions of this article are as follows:

- the description of a theoretical framework based on ProMPs for predicting the human desired trajectory and goal during physical human–robot interaction, providing the following features: recognition of the current task, estimation of the task duration, and prediction of the future trajectory;
- an experimental study about how multimodal information can be used to improve the estimation of the duration/speed of an initiated trajectory;
- the open-source software to realize an intention recognition application with the iCub robot, both in simulation and on the real robot.

The article is organized as follows. In Section 2, we review the literature about intentions in Human–Robot Interaction (HRI), probabilistic models for motion primitives, and their related software. In Section 3, we describe the theoretical tools that we use to formalize the problem of predicting the intention of the human during interaction. Particularly, we describe the ProMPs and their use for predicting the evolution of a trajectory given early observations. In Section 4, we overview the software organization and the interconnection between our software and the iCub's main software, both for the real and simulated robot. The following sections are devoted to presenting our software and its use for predicting intention. We choose to present three examples of increasing complexity, with the simulated and real robot. We provide and explain in detail a software example for a 1-DOF trajectory in Section 5. In Sections 6 and 7, we present the intention recognition application with the simulated and real iCub, respectively. In the first examples with the robot, the "tasks" are exemplified by simple reaching movements, to provide simple and clear trajectories that help the reader understand the method, whereas the last experiment with the robot is a collaborative object sorting task. Section 8 provides the links to the videos showing how to use the software in simulation and on the iCub. Finally, in Section 10, we discuss our approach and its limitations and outline our future developments.

2. RELATED WORK

In this article, we propose a method to recognize the intention of the human partner collaborating with the robot, formalized as the target and the "future" trajectory associated with a skill, modeled by a goal-directed Probabilistic Movement Primitive. In this section, we briefly overview the literature about intention recognition in human–robot interaction and motion primitives for learning of goal-directed robotic skills.

2.1. Intention during Human–Robot Interaction

When humans and robots collaborate, mutual understanding is paramount for the success of any shared task. Mutual understanding means that the human is aware of the robot's current task, status, goal, available information, that he/she can reasonably predict or expect what it will do next, and *vice versa*. Recognizing the intention is only one piece of the problem but still plays a crucial part for providing anticipatory capabilities.

Formalizing intention can be a daunting task, as one may find it difficult to provide a unique representation that explains the intention for very low-level goal-directed tasks (e.g., reaching a target object and grasping it) and for very high-level, complex, abstract or cognitive tasks (e.g., change a light bulb on the ceiling—by building a stair composed of many parts, climbing it and reaching the light bulb on the ceiling, etc.). Demiris (2007) reviews different approaches of action recognition and intention prediction.

From the human's point of view, understanding the robot's intention means that the human should find intuitive and non-ambiguous every goal-directed robot movement or actions, and it should be clear what the robot is doing or going to do (Kim et al., 2017). Dragan and Srinivasa (2014) formalized the difference between *predictability* and *legibility*: a motion is legible if an observer can quickly infer its goal, while a motion is predictable when it matches the expectations of the observer given its goal.

The problem of generating *legible* motions for robots has been addressed in many recent works. For example, Dragan and Srinivasa (2014) use optimization techniques to generate movements that are predictable and legible. Huang et al. (2017) apply an Inverse Reinforcement Learning method on autonomous cars to select the robot movements that are maximally informative for the humans and that will facilitate their inference of the robot's objectives.

From the robot's point of view, understanding the human's intention means that the robot should be able to decipher the ensemble of verbal and non-verbal cues that the human naturally generates with his/her behavior, to identify, for a current task and context, what is the human intention. The more information (e.g., measurable signals from the human and the environment) is used, the better and more complex the estimation can be.

The simplest form of intention recognition is to estimate the goal of the current action, under the implicit assumption that each action is a goal-directed movement.

Sciutti et al. (2013) showed that humans implicitly attribute intentions in form of goals to robot motions, proving that humans exhibit anticipatory gaze toward the intended goal. Gaze was also used by Ivaldi et al. (2014a) in a human–robot interaction game with iCub, where the robot (human) was tracking the human (robot) gaze to identify the target object. Ferrer and Sanfeliu (2014) proposed the Bayesian Human Motion Intentionality Prediction algorithm, to geometrically compute the most likely target of the human motion, using Expectation–Maximization and a simple Bayesian classifier. In Wang et al. (2012), a method called Intention-Driven Dynamics model, based on Gaussian Process Dynamical Models (GPDM) (Wang et al., 2005), is used to infer the intention of the robot's partner during a ping-pong match, represented by the target of the ball, by analyzing the entire human movement before the human hits the ball. More generally, modeling and descriptive approaches can be used to match predefined labels with measured data (Csibra and Gergely, 2007).

A more complex form of intention recognition is to estimate the future trajectory from the past observations. In a sense, to estimate $[x_{t+1}, \ldots, x_{t+T_{future}}] = f(x_t, x_{t-1}, \ldots, x_{t-T_{past}})$. This problem, very similar to the estimate of the forward dynamics model of a system, is frequently addressed by researchers in model predictive control, where being able to "play" the system evolving in time is the basis for computing appropriate robot controls. When a trajectory can be predicted by an observer from early observations of it, we can say that the trajectory is not only *legible*, but *predictable*. A systematic approach for predicting a trajectory is to reason in terms of movement primitives, in such a way that the sequence of points of the trajectory can be generated by a parametrized time model or a parametrized dynamical system. For example, Palinko et al. (2014) plan reaching trajectories for object carrying that are able to convey information about the weight of the transported object. More generally, in generative approaches (Buxton, 2003), latent variables are used to learn models for the primitives, both to generate and infer actions. The next subsection will provide more detail about the state-of-the-art techniques for generating movement primitives.

In Amor et al. (2014), the robot first learns Interaction Primitives by watching two humans performing an interactive task, using motion capture. The Interaction Primitive encapsulates the dependencies between the two human movements. Then, the robot uses the Interaction Primitive to adapt its behavior to its partner's movement. Their method is based on Dynamics Motor Primitives (Ijspeert et al., 2013), where a distribution over the DMP's parameters is learned. Notably, in this article, we did not follow the same approach to learn Interaction Primitives, since there is a physical interaction that makes the user's and the robot's movements as one joint movements. Moreover, there is no latency between the partner's early movement and the robot's, because the robot's arm is physically driven by the human until the latter breaks the contact.

Indeed, most examples in the literature focus on kinematic trajectories, corresponding to gestures that are typically used in dyadic interactions characterized by a coordination of actions and reactions. Whenever the human and robot are also interacting physically, collaborating on a task with some exchange of forces, then the problem of intention recognition becomes more complex. Indeed, the kinematics information provided by the "trajectories" cannot be analyzed without taking into account the haptic exchange and the estimation of the "roles" of the partners in leading/following each other.

Estimating the current role of the human (master/slave or leader/follower) is crucial, as the role information is necessary to coherently adapt the robot's compliance and impedance at the level of the exchanged contact forces. Most importantly, adapting the haptic interaction can be used by the robot to communicate when it has understood the human intent and is able to finish the task autonomously, mimicking the same type of implicit non-verbal communication that is typical of humans.

For example, in Gribovskaya et al. (2011), the robot infers the human intention utilizing the measure of the human's forces and by using Gaussian Mixture Models. In Rozo Castañeda et al. (2013), the arm impedance is adapted by a Gaussian Mixture Model based on measured forces and visual information. Many studies focused on the robot's ability to act only when and how its user wants (Carlson and Demiris, 2008; Soh and Demiris, 2015) and to not interfere with the partner's forces (Jarrassé et al., 2008) or actions (Baraglia et al., 2016).

In this article, we describe our approach to the problem of recognizing the human intention during collaboration by providing an estimate of the future intended trajectory to be performed by the robot. In our experiments, the robot does not adapt its role during the physical interaction but simply switches from follower to leader when the human breaks contact with it.

2.2. Movement Primitives

Movement Primitives (MPs) are a well established paradigm for representing complex motor skills. The most known method for representing movement primitives is probably the Dynamic Movement Primitives (DMPs) (Schaal, 2006; Ijspeert et al., 2013; Meier and Schaal, 2016). DMPs use a stable non-linear attractor in combination with a forcing term to represent the movement. The forcing term enables to follow specific movement, while the attractor asserts asymptotic stability. In a recent paper, Meier and Schaal (2016) proposed an extension to DMPs, called PDMP (Probabilistic Dynamic Movement Primitive). This method improves DMP with probabilistic properties to measure the likelihood that the movement primitive is executed correctly and to perform inference on sensor measurement. However, the PDMPs do not have a data-driven generalization and can deviate arbitrarily from the demonstrations. This last difference can be critical for our applications with the humanoid robot iCub, since uncertainties are unavoidable and disturbances may happen frequently and destabilize the robot movement (for example, an unexpected collision during the movement). Thus, the ProMPs method is more accurate for our software.

Ewerton et al. (2015), Paraschos et al. (2013b), and Maeda et al. (2014) compared ProMPs and DMPs for learning primitives and specifically interaction primitives. With the DMP model, at the end of the movement, only a dynamic attractor is activated. Thus, it always reaches a stable goal. The properties allowed by both methods are temporal scaling of the movement, learning from a single demonstration, and generalizing to new final position. With ProMPs, we have in addition the ability to do inference (thanks to the distribution), to force the robot to pass by several initial via points (the early observations), to know the correlation between the input of the model, and to co-activate some ProMPs. In our study, we need these features, because the robot must determine a trajectory that passes by the early observations (beginning of the movement where the user guides physically the robot).

A Recurrent Neural Networks (RNN) approach (Billard and Mataric, 2001) used a hierarchy of neural networks to simulate the activation of areas in human brain. The network can be trained to infer the state of the robot at the next point in time, given the current state. The authors propose to train the RNN by minimizing the error between the inferred position of the next time step and the ground truth obtained from demonstrations.

Hidden Markov Models (HMMs) for movement skills were introduced by Fine et al. (1998). This method is often used to categorize movements, where a category represents a movement primitive. This method also allows to represent the temporal sequence of a movement. In Nguyen et al. (2005), they use learned Hierarchical Hidden Markov Model (HHMMs) to recognize human behaviors efficiently. In Ren and Xu (2002), they present the Primitive-based Coupled-HMM (CHMM) approach, for human natural complex action recognition. In this approach, each primitive is represented by a Gaussian Mixture Model.

Adapting Gaussian Mixture Models is another method used to learn physical interaction with learning. In Evrard et al. (2009), they use GMMs and Gaussian Mixture Regression to learn, in addition to the position (joint information), force information. Using this method, a humanoid robot is able to collaborate in one dimension with its partner for a lifting task. In this article, we will also use (Cartesian) position and force information to allow our robot to interact physically with its partner.

A subproblem of movement recognition is that robots need to estimate the duration of the trajectory to align a current trajectory with learned movements. In our case, at the beginning of the physical Human–Robot Interaction (pHRI), the robot observes a partial movement guided by its user. Given this partial movement, the robot must first estimate what the current state of the movement is to understand what its partner intent is. Thus, it needs to estimate the partial movement's speed.

Fitts' law models the movement duration for goal-directed movements. This model is based on the assumption that the movement duration is a linear function of the difficulty to achieve a target (Fitts, 1992). In Langolf et al. (1976), they show that by modifying the target's width, the shape of the movement changes. Thus, it is difficult to apply Fitt's law when the size of the target can change. In Langolf et al. (1976) and Soechting (1984), they confirm this idea by showing that the shape of the movement changes with the accuracy required by the goal position of the movement.

Dynamics Time Warping (DTW) is a method to find the correlation between two trajectories that have different durations, in a more robust way than the Euclidean distance. In Amor et al. (2014), they modify the DTW algorithm to match a partial movement with a reference movement. Many improvements over this method exist. In Keogh (2002), they propose a robust method to improve the indexation. The calculation speed of DTW is improved using different methods, such as FastDTW, Lucky Time Warping, or FTW. An explanation and comparison of these methods are presented in Silva and Batista (2016), where they add their own computation speed improvement by using a method called Pruned Warping Paths. This method allows the deletion of unlikely data. However, a drawback of this well-known DTW method is they do not preserve the global trajectory's shape.

In Maeda et al. (2014), where they use a probabilistic learning of movement primitives, they improve the duration estimation of movements by using a different time warping method. This method is based on a Gaussian basis model to represent a time warping function and, instead of DTW, it forces a local alignment between the two movements without "jumping" some index. Thus, the resulting trajectories are more realistic, smoother, and this method preserves the global trajectories' shapes.

For inferring the intention of the robot's partner, we use Probabilistic Movement Primitives (ProMPs) (Paraschos et al., 2013a). Specifically, we use the ProMP's conditioning operator to adapt the learned skills according to observations. The ProMPs can encode the correlations between forces and positions and allow better prediction of the partner's intention. Further, the phase of the partner's movement can be inferred, and therefore the robot can adapt to the partner's velocity changes. ProMPs are more efficient for collaborative tasks, as shown in Maeda et al. (2014), where in comparison to DMPs, the root-mean square error of the predictions is lower.

2.3. Related Open-Source Software

One of the goals of this article is to introduce an open-source software for the iCub (but potentially for any other robot), where the ProMP method is used to recognize human intention during collaboration, so that the robot can execute initiated actions autonomously. This is not the first open-source implementation for representing movement primitives: however, it has a novel application and a rationale that makes it easy to use with the iCub robot.

In **Table 1**, we report on the main software libraries that one can use to learn movement primitives. Some have been also used to realize learning applications with iCub, e.g., Lober et al. (2014) and Stulp et al. (2013) or to recognize human intention. However, the software we propose here is different: it provides an implementation of ProMPs used explicitly for intention recognition and prediction of intended trajectories. It is interfaced with iCub, both

TABLE 1 | Open-source software libraries implementing movement primitives and their application to different known robots.

Software/library	Method	Code link	Language	Robot	Reference
Dynamical System Modulation for Robot Adaptive Learning via Kinesthetic Demonstrations	GMR	Hersch et al. (2008)	Matlab	Hoap3	Micha and Aude (2008)
pbdlib-matlab	HMM, GMM, and others	Calinon (2015)	Matlab	Baxter	Calinon (2016)
DMP learning with GMR	DMP and GMR	Calinon et al. (2012a)	Matlab or C	Coman	Calinon et al. (2012b)
Stochastic Machine Learning Toolbox	Kernel Functions, Gaussian Processes, Bayesian Optimization	Lober (2014)	C++ or Python	–	
pydmps	DMP	DeWolf (2013)	Python	Sarcos	Ijspeert et al. (2013)
Dynamical Systems approach to Learn Robot Motions	GMM and SEDS	Khansari (2011)	Matlab	iCub	Khansari-Zadeh and Billard (2011, 2012)
Function Approximation, DMP, and Black-Box Optimization (dmpbbo)	DMP	Stulp (2014)	Python or C++	iCub	Stulp et al. (2013), Lober et al. (2014)
Learning Motor Skills from Partially Observed Movements Executed at Different Speeds	ProMP	Ewerton (2016)	Matlab or Python	–	Ewerton et al. (2015)
icubLearningTrajectories	ProMP	Dermy (2017)	Matlab and C++	iCub	–

real and simulated, and addresses in the specific case of physical interaction between the human and the robot. In short, it is a first step toward adding intention recognition ability to the iCub robot.

3. THEORETICAL FRAMEWORK

In this section, we present the theoretical framework that we use to tackle the problem of intent recognition: we describe the ProMPs and how they can be used to predict trajectories from early observations.

In Section 2, we formulate the problem of learning a primitive for a simple case, where the robot learns the distribution from several demonstrated trajectories. In Section 3.3, we formulate and provide the solution to the problem of predicting the "future" trajectory from early observations (i.e., the initial data points). In Section 3.4, we discuss the problem of predicting the time modulation, i.e., predicting the global duration of the predicted trajectory. This problem is non-trivial, as by construction the demonstrated trajectories are "normalized" in duration when the ProMP is learned.[7] In Section 3.5, we explain how to recognize, from the early observations, to which of many known skills (modeled by ProMPs) the current trajectory belongs. In all these sections, we tried to present the theoretical aspects related to the use of ProMPs for the intention recognition application.

Practical examples of these theoretical problems are presented and explained later in sections 5–7. Section 5 explains how to use our software, introduced in Section 4, for learning one ProMP for a simple set of 1-DOF trajectories. Section 6 presents an example with the simulated iCub in Gazebo, while Section 7 presents an example with the real iCub.

3.1. Notation

To facilitate understanding of the theoretical framework, we first introduce the notations we use in this section and throughout the remainder of the article.

3.1.1. Trajectories

- $X(t) \in \mathbb{R}^3$, $X(t) = [x(t), y(t), z(t)]^\mathrm{T}$: the x/y/z-axis Cartesian coordinate of the robot's end effector.
- $F(t) \in \mathbb{R}^6$, $F(t) = [f_x, f_y, f_z, m_x, m_y, m_z]^\mathrm{T}$: the wrench contact forces, i.e., the external forces and moments measured by the robot at the contact level (end effector).
- $\xi(t) \in \mathbb{R}^D$: the generic vector containing the current value or state of the trajectories at time t. It can be monodimensional (e.g., $\xi(t) = [z(t)]$), or multidimensional (e.g., $\xi(t) = [X(t), F(t)]^\mathrm{T}$), depending on the type of trajectories that we want to represent with the ProMP.
- $\Xi = \Xi_{[1:t_f]} = [\xi(1), \ldots, \xi(t_f)]^\mathrm{T} \in \mathbb{R}^{D \cdot t_f}$ is an entire trajectory, consisting of t_f samples or data points.
- $\Xi_{i[1:t_{fi}]}$ is the i-th demonstration (trajectory) of a task, consisting of t_{fi} samples or data points.

3.1.2. Movement Primitives

- $k \in [1:K]$: the k-th ProMP, among a set of K ProMPs that represent different tasks/actions.
- n_k: number of recorded trajectories for each ProMP.
- $S_k = \{\Xi_{\{k,1\}}, \ldots, \Xi_{\{k,n_k\}}\}$: set of n_k trajectories for the k-th ProMP.
- $\xi(t) = \Phi_t \boldsymbol{\omega} + \epsilon_\xi$ is the model of the trajectory with:
  - $\epsilon_\xi \sim \mathcal{N}(0, \beta)$: expected trajectory noise.
  - $\Phi_t \in \mathbb{R}^{D \times D \cdot M}$: radial basis functions (RBFs) used to model trajectories. It is a block diagonal matrix.
    - M: number of RBFs.
    - $\psi_{ji}(t) = \dfrac{e^{\frac{-(t-c_i)^2}{2h}}}{\sum_{m=1}^{M} e^{\frac{-(t-c_m)^2}{2h}}}$: i-th RBF for all inputs $j \in [1:D]$.

    It must be noted that the upper term comes from a Gaussian $\frac{1}{\sqrt{2\pi h}} e^{\frac{-(t-c_i)^2}{2h}}$, where c_i and h are, respectively, the center and variance of the i-th Gaussian. In our RBF formulation, we normalize all the Gaussians.
  - $\boldsymbol{\omega} \in \mathbb{R}^{D \cdot M}$: time-independent parameter vector weighting the RBFs, i.e., the parameters to be learned.
- $p(\boldsymbol{\omega}) \sim \mathcal{N}(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$: normal distribution computed from a set $\{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n\}$. It represents the distribution of the modeled trajectories, also called *prior* distribution.

3.1.3. Time Modulation

- $\bar{s}$: number of samples used as reference to rescale all the trajectories to the same duration.
- $\Phi_{\alpha_i t} \in \mathbb{R}^{D \times D \cdot M}$: the RBFs rescaled to match the Ξ_i trajectory duration.
- $\alpha_i = \frac{\bar{s}}{t_{fi}}$: temporal modulation parameter of the i-th trajectory.
- $\alpha = \Psi_{\delta_{n_o}} \boldsymbol{\omega}_\alpha + \epsilon_\alpha$ is the model of the function mapping δ_{n_o} into the temporal modulation parameter α, with:
  - Ψ: a set of RBFs used to model the mapping between δ_{n_o} and α;
  - δ_{n_o} is the variation of the trajectory during the first n_o observations (data points); it can be $\delta_{n_o} = \xi(n_o) - \xi(1)$ if the entire trajectory variables (e.g., Cartesian position and forces) are considered, or more simply $\delta_{n_o} = X(n_o) - X(1)$ if only the variation in terms of Cartesian position is considered;
  - $\boldsymbol{\omega}_\alpha$: the parameter vector weighting the RBFs of the Ψ matrix.

3.1.4. Inference

- $\Xi^o = [X^o, F^o]^\mathrm{T} = [\xi^o(1), \ldots, \xi^o(n_o)]^\mathrm{T}$: early-trajectory observations, composed of n_o data points.
- Σ^o_ξ: noise of the initiated trajectory observation.
- $\hat{\alpha}$: estimated time modulation parameter of a trajectory to infer.
- $\hat{t}_f = \frac{\bar{s}}{\hat{\alpha}}$: estimated duration of a trajectory to infer.
- $\Xi^* = [\xi^o(1), \ldots, \xi^o(n_o), \xi^*(n_o+1), \ldots, \xi^*(t_f)]$: ground truth of the trajectory for the robot to infer.
- $\hat{\Xi} = [\hat{X}, \hat{F}]^\mathrm{T} = [\xi^o(1), \ldots, \xi^o(n_o), \hat{\xi}(n_o+1), \ldots, \hat{\xi}(\hat{t}_f)]^\mathrm{T}$: the estimated trajectory.
- $p(\hat{\boldsymbol{\omega}}) \sim \mathcal{N}(\hat{\mu}_{\boldsymbol{\omega}}, \hat{\sigma}_{\boldsymbol{\omega}})$: posterior distribution of the parameter vector of a ProMP using the observation Ξ^o.
- $\hat{k}$: index of the recognized ProMP from the set of K known (previously learned) ProMPs.

[7] In some tasks, *e.g.,* reaching, it is reasonable to assume that the difference of duration of the demonstrated trajectories is negligible; however, in other tasks, the duration of the demonstrated trajectories may vary significantly.

3.2. Learning a Probabilistic Movement Primitive (ProMP) from Demonstrations

Our toolbox to learn, replay and infer the continuation of trajectories is written in Matlab and available at:

https://github.com/inria-larsen/icubLearningTrajectories/tree/master/MatlabProgram

Let us assume the robot has recorded a set of n_1 trajectories: $\{\Xi_1, \ldots, \Xi_{n_1}\}$, where the i-th trajectory is $\Xi_i = \{\xi(1), \ldots, \xi(t_{f_i})\}$. $\xi(t)$ is the generic vector containing all the variables to be learned at time t, with the ProMP method. It can be monodimensional (e.g., $\xi(t) = [z(t)]$ for the z-axis Cartesian coordinate), or multi-dimensional (e.g., $\xi(t) = [X(t), F(t)]^{\mathrm{T}}$). Note that the duration of each recorded trajectory (i.e., t_{f_i}) may be variable. To find a common representation in terms of primitives, a time modulation is applied to all trajectories, such that they have the same number of samples $\bar{s}$ (see details in Section 3.4). Such modulated trajectories are then used to learn a ProMP.

A ProMP is a Bayesian parametric model of the demonstrated trajectories in the following form:

$$\xi(t) = \Phi_t \boldsymbol{\omega} + \epsilon_\xi, \tag{1}$$

where $\boldsymbol{\omega} \in R^M$ is the time-independent parameter vector weighting the RBFs, $\epsilon_\xi \sim \mathcal{N}(0, \beta)$ is the trajectory noise, and Φ_t is a vector of M radial basis functions evaluated at time t:

$$\Phi_t = [\psi_1(t), \psi_2(t), \ldots, \psi_M(t)]$$

with

$$\begin{cases} \psi_i(t) &= \frac{1}{\sum_{j=1}^{M} \psi_j(t)} exp\left\{ \frac{-(t-c(i))^2}{2h} \right\} \\ c(i) &= i/M \\ h &= 1/M^2. \end{cases} \tag{2}$$

Note that all the ψ functions are scattered across time.

For each Ξ_i trajectory, we compute the ω_i parameter vector to have $\xi_i(t) = \Phi_t \omega_i + \epsilon_\xi$. This vector is computed to minimize the error between the observed $\xi_i(t)$ trajectory and its model $\Phi_t \omega_i + \epsilon_\xi$. This is done using the Least Mean Square algorithm, i.e.:

$$\omega_i = (\Phi_t^\top \Phi_t)^{-1} \Phi_t^\top \xi_i(t). \tag{3}$$

To avoid the common issue of the matrix $\Phi_t^{\mathrm{T}} \Phi_t$ in equation (3) not being invertible, we add a diagonal term and perform Ridge Regression:

$$\omega_i = (\Phi_t^\top \Phi_t + \lambda)^{-1} \Phi_t^\top \xi_i(t), \tag{4}$$

where $\lambda = 10^{-11} \cdot 1_{D \cdot M \times D \cdot M}$ is a parameter that can be tuned by looking at the smallest singular value of the matrix $\Phi_t^{\mathrm{T}} \Phi_t$.

Thus, we obtain a set of these parameters: $\{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n\}$, upon which a distribution is computed. Since we assume Normal distributions, we have:

$$p(\boldsymbol{\omega}) \sim \mathcal{N}(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}) \tag{5}$$

$$\text{with } \mu_{\boldsymbol{\omega}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\omega_i} \tag{6}$$

$$\text{and } \Sigma_{\boldsymbol{\omega}} = \frac{1}{n-1} \sum_{i=1}^{n} (\boldsymbol{\omega_i} - \mu_{\boldsymbol{\omega}})^\top (\boldsymbol{\omega_i} - \mu_{\boldsymbol{\omega}}). \tag{7}$$

The ProMP captures the distribution over the observed trajectories. To represent this movement primitive, we usually use the movement that passes by the mean of the distribution. **Figure 2** shows the ProMP for a 1-DOF lifting motion, with a number of reference samples $\bar{s} = 100$ and number of basis functions $M = 5$.

This example is included in our Matlab toolbox as `demo_plot1DOF.m`. The explanation of this Matlab script is presented in Section 5. More complex examples are also included in the scripts `demo_plot*.m`.

3.3. Predicting the Future Movement from Initial Observations

Once the ProMP $p(\boldsymbol{\omega}) \sim \mathcal{N}(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$ of a certain task has been learned,[8] we can use it to predict the evolution of an initiated movement. An underlying hypothesis is that the observed movement follows to this learned distribution.

[8] That is, we computed the $p(\boldsymbol{\omega})$ distribution from the dataset $\{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n\}$, where each $\boldsymbol{\omega}_i$ is an estimated parameter computed from the trajectory demonstrations.

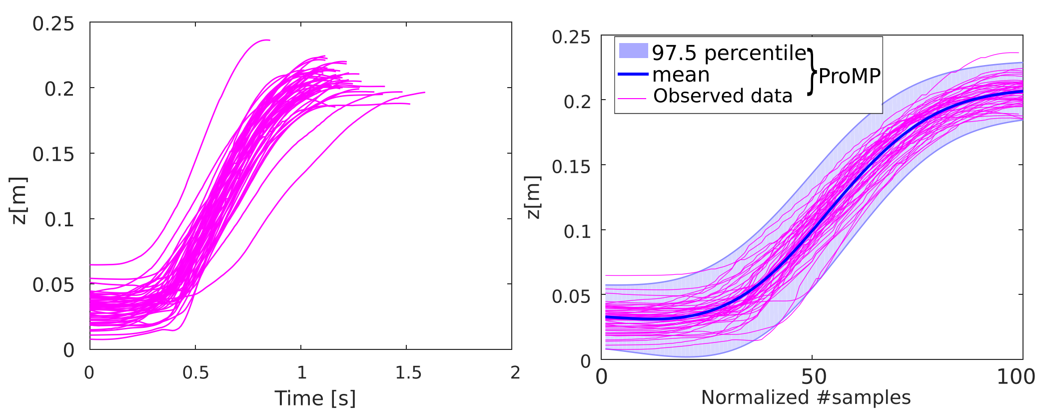

FIGURE 2 | The observed trajectories are represented in magenta. The corresponding ProMP is represented in blue. The following parameters are used: $\bar{s} = 100$ for the reference number of samples, $M = 5$ for the number of RBFs spread over time, and $h = 0.04$ $\left(= \frac{1}{M^2}\right)$ the variance of the RBFs.

Suppose that the robot measures the first n_o observations of the trajectory to predict (e.g., lifting the arm). We call these observations $\Xi^o = [\xi^o(1), \ldots, \xi^o(n_o)]$. The goal is then to predict the evolution of the trajectory after these n_o observations, i.e., find $\{\hat{\xi}(n_o+1), \ldots, \hat{\xi}(\hat{t}_f)\}$, where $\hat{t}_f$ is the estimation of the trajectory duration (see Section 3.4). This is equivalent to predicting the entire $\hat{\Xi}$ trajectory where the first n_o samples are known and equal to the observations: $\hat{\Xi} = \{\xi^o(1), \ldots, \xi^o(n_o), \hat{\xi}(n_o+1), \ldots, \hat{\xi}(t_{\hat{t}_f})\}$. Therefore, our prediction problem consists of predicting $\hat{\Xi}$ given the Ξ^o observations.

To do this prediction, we start from the learned prior distribution $p(\boldsymbol{\omega})$, and we find the $\hat{\boldsymbol{\omega}}$ parameter within this distribution that generates $\hat{\Xi}$. To find this $\hat{\boldsymbol{\omega}}$ parameter, we update the learned distribution $p(\hat{\boldsymbol{\omega}}) \sim \mathcal{N}(\hat{\mu}_{\boldsymbol{\omega}}, \hat{\Sigma}_{\boldsymbol{\omega}})$ using the following formulae:

$$\begin{cases} \hat{\mu}_{\boldsymbol{\omega}} = \mu_{\boldsymbol{\omega}} + K(\Xi^o - \Phi_{[1:n_o]}\, \mu_{\boldsymbol{\omega}}) \\ \hat{\Sigma}_{\boldsymbol{\omega}} = \Sigma_{\boldsymbol{\omega}} - K(\Phi_{[1:n_o]}\, \Sigma_{\boldsymbol{\omega}}), \end{cases} \tag{8}$$

where K is a gain computed by the following equation:

$$K = \Sigma_{\boldsymbol{\omega}} \Phi_{[1:n_o]}^{\top} \left(\Sigma_{\xi}^{o} + \Phi_{[1:n_o]} \Sigma_{\boldsymbol{\omega}} \Phi_{[1:n_o]}^{\top} \right)^{-1}. \tag{9}$$

Equations (8) and (9) can be computed through the marginal and conditional distributions (Bishop, 2006; Paraschos et al., 2013a), as detailed in Appendix A.

Figure 3 shows the predicted trajectory for the lifting motion of the left arm of iCub. The different graphs show inferred trajectories when the robot observed $n_o = 10$, 30, 50, and 80% of the total trajectory duration. This example is also available in the toolbox as `demo_plot1DOF.m`. The `nbData` variable changes the percentage of known data. Thus, it will be visible how the inference improves according to this variable. An example of predicted trajectories of the arm lifting in Gazebo can be found in a provided video (see Section 8).

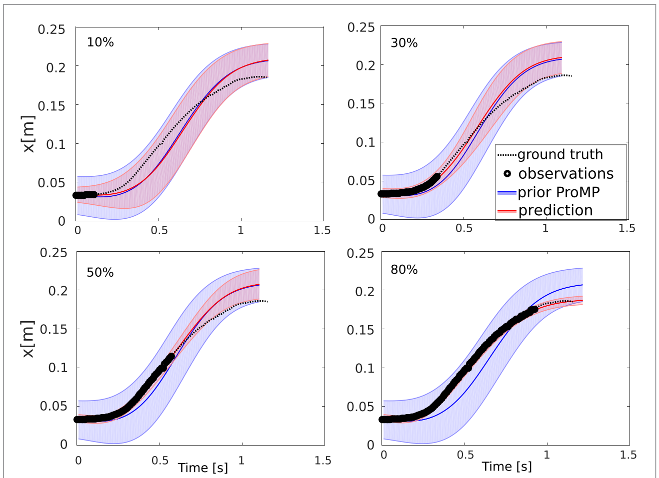

FIGURE 3 | The prediction of the future trajectory given early observations, exploiting the information of the learned ProMP (**Figure 2**). The plots show the predicted trajectories after 10, 30, 50, and 80% of observed data points.

3.4. Predicting the Trajectory Time Modulation

In the previous section, we presented the general formulation of ProMPs, which makes the implicit assumption that all the observed trajectories have the same duration and thus the same sampling.[9] That is why the duration of the trajectories generated by the RBF is fixed and equal to $\bar{s}$. Of course, this is valid only for synthetic data and not for real data.

To be able to address real experimental conditions, we now consider the variation of the duration of the demonstrated trajectories. To this end, we introduce a time modulation parameter α that maps the actual trajectory duration t_f to $\bar{s}$: $\alpha = \bar{s}/t_f$. The normalized duration $\bar{s}$ can be chosen arbitrarily; for example it can be set to the average of the duration of the trajectories, e.g., $\bar{s} = mean(t_{f1}, \ldots, t_{fK})$. Notably, in the literature sometimes α is called *phase* (Paraschos et al., 2013a,b). The effect of α is to change the phase of the RBFs, which are scaled in time.

The time modulation of the i-th trajectory Ξ_i is computed by $\alpha_i = \frac{\bar{s}}{t_{fi}}$. Thus, we have $\alpha \cdot t \in [1 : \bar{s}]$. Thus, the improved ProMP model is as follows:

$$\xi_t = \Phi_{\alpha t} \boldsymbol{\omega} + \epsilon_t, \tag{10}$$

where $\Phi_{\alpha t}$ is the RBFs matrix evaluated at time αt. All the M Gaussian functions of the RBFs are spread over the same number of samples $\bar{s}$. Thus, we have the following:

$$\Phi_{\alpha t} = [\psi_1(\alpha t), \psi_2(\alpha t), \ldots, \psi_M(\alpha t)].$$

During the learning step, we record a set of α parameters: $S_\alpha = \{\alpha_1, \ldots, \alpha_n\}$. Then, using this set, we can replay the learned ProMP with different speeds. By default (e.g., when $\alpha = 1$), the speed allows to finish the movement in $\bar{s}$ samples.

During the inference, the time modulation α of the partially observed trajectory is not known. Unless fixed *a priori*, the robot must estimate it. This estimation is critical to ensure a good recognition, as shown in **Figure 4**: the inferred trajectory (represented by the mean of the posterior distribution in red) does not have the same duration as the "real" intended trajectory (which is the ground truth). This difference is due to the estimation error of the time modulation parameter. This estimation $\hat{\alpha}$ by default is computed as the mean of all the α_k observed during the learning:

$$\hat{\alpha} = \frac{\sum \alpha_k}{n_k}. \tag{11}$$

However, using the mean value for the time modulation is an appropriate choice only when the primitive represents goal-directed motions that are very regular, or for which we can reasonably assume that differences in the duration can be neglected (which is not a general case). In many applications, this estimation may be too rough.

Thus, we have to find a way to estimate the duration of the observed trajectory, which corresponds to accurately estimating the time modulation parameter $\hat{\alpha}$. To estimate $\hat{\alpha}$, we implemented

[9] Actually, we call here duration what is in fact the total number of samples for the trajectory.

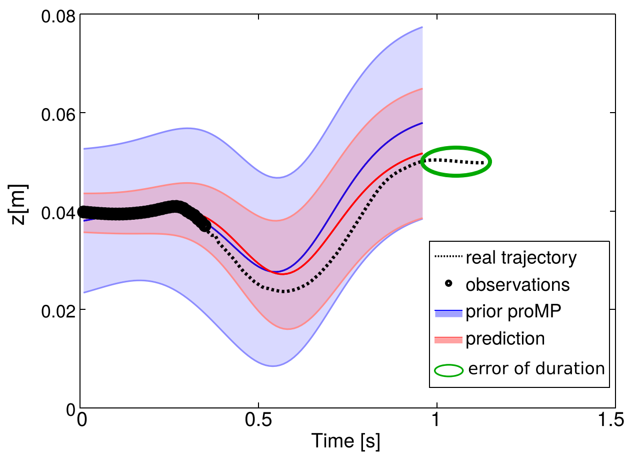

FIGURE 4 | This plot shows the predicted trajectory given early observations (data points, in black), compared to the ground truth (e.g., the trajectory that the human intends to execute with the robot). We show the prior distribution (in light blue) and the posterior distribution (in red), which is computed by conditioning the distribution to match the observations. Here, the posterior simply uses the average α computed over the $\alpha_1, \ldots, \alpha_K$ of the K demonstrations. Without predicting the time modulation from the observations and using the average α, the predicted trajectory has a duration that is visibly different from the ground truth.

four different methods. The first is the **mean of all the** α_k, as in equation (11). The second is the **maximum likelihood**, with

$$\hat{\alpha} = \mathrm{argmax}_{\alpha \in S_{\alpha k}} \{\mathrm{loglikelihood}(\Xi^o, \mu_{\boldsymbol{\omega}_k}, \sigma_{\boldsymbol{\omega}_k}, \alpha_k)\}. \quad (12)$$

The third is the **minimum distance** criterion, where we seek the best $\hat{\alpha}$ that minimizes the difference between the observed trajectory Ξ_t^o and the predicted trajectory for the first n_o data points:

$$\hat{\alpha} = \mathrm{argmin}_{\alpha \in S_{\alpha k}} \left\{ \sum_{t=1}^{n_o} |\Xi_t^o - \Phi_{\alpha t} \mu_{\boldsymbol{\omega}_k}| \right\}. \quad (13)$$

The fourth method is based on a **model**: we assume that there is a correlation between α and the variation of the trajectory δ_{n_o} from the beginning until the time n_o. This "variation" δ_{n_o} can be computed as the variation of the position, e.g., $\delta_{n_o} = X(n_o) - X(1)$, or the variation in the entire trajectory, $\delta_{n_o} = \Xi(n_o) - \Xi(1)$, or any other measure of progress, if this hypothesis is appropriate for the type of task trajectories of the application.[10] Indeed, the α can be linked also to the movement speed, which can be roughly approximated by $\dot{X} = \frac{\delta X}{t_f}$ $\left(\dot{\Xi} = \frac{\delta \Xi}{t_f}\right)$. We model the mapping between δ_{n_o} and α by the following equation:

$$\alpha = \Psi(\delta_{n_o})^\top \boldsymbol{\omega}_\alpha + \epsilon_\alpha, \quad (14)$$

where Ψ are RBFs, and ϵ_α is a zero-mean Gaussian noise. During learning, we compute the $\boldsymbol{\omega}_\alpha$ parameter, using the same method as in equation (3). During the inference, we compute $\hat{\alpha} = \Psi(\delta_{n_o})^{\mathrm{T}} \boldsymbol{\omega}_\alpha$.

[10] In our case, this assumption can be appropriate, because the reaching trajectories in our application are generally monotonic increasing/decreasing.

A comparison of the four methods for estimating α on a test study with iCub in simulation is presented in Section 6.6.

There exist other methods in the literature for computing α. For example, Ewerton et al. (2015) propose a method that models local variability in the speed of execution. In Maeda et al. (2016), they use a method that improves Dynamic Time Warping by imposing a smooth function on the time alignment mapping using local optimization. These methods will be implemented in the future works.

3.5. Recognizing One among Many Movement Primitives

Robots should not learn only one skills but many: different skills for different tasks. In our framework, tasks are represented by movement primitives, precisely ProMP. So it is important for the robot to be able to learn K different ProMPs and then be able to recognize from the early observations of a trajectory which of the K ProMPs the observations belong to.

During the learning step of a movement primitive $k \in [1 : K]$, the robot observes different trajectories $S_k = \{\Xi_1, \ldots, \Xi_n\}$. For each ProMP, it learns the distribution over the parameters vector $p(\boldsymbol{\omega}) \sim \mathcal{N}(\mu_{\boldsymbol{\omega}_k}, \Sigma_{\boldsymbol{\omega}_k})$, using equation (3). Moreover, the robot records the different phases of all the observed trajectories: $S_{\alpha k} = \{\alpha_{1k}, \ldots, \alpha_{nk}\}$.

After having learned these K ProMPs, the robot can use this information to autonomously execute a task trajectory. Since we are targeting collaborative movements, performed together with a partner at least at the beginning, we want the robot to be able to recognize from the first observations of a collaborative trajectory which is the current task that the partner is doing and what is the intention of the partner. Finally, we want the robot to be able to complete the task on its own, once it has recognized the task and predicted the future trajectory.

Let $\Xi^o = [\Xi_1 \ldots \Xi_{n_o}]^{\mathrm{T}}$ be the early observations of an initiated trajectory.

From these partial observations, the robot can recognize the "correct" (i.e., most likely) ProMP $\hat{k} \in [1 : K]$. First, for each ProMP $k \in [1 : K]$, it computes the most likely phase (time modulation factor) $\hat{\alpha}_k$ (as explained in Section 3.4), to obtain the set of ProMPs with the most likely duration: $S_{[\mu_{\omega_k}, \hat{\alpha}_k]} = \{(\mu_{\boldsymbol{\omega}_1}, \hat{\alpha}_1), \ldots, (\mu_{\boldsymbol{\omega}_K}, \hat{\alpha}_K)\}$.

Then we compute the most likely ProMP $\hat{k}$ in $S_{[\mu_{\omega_k}, \hat{\alpha}_k]}$ according to some criterion. One possible way is to minimize the distance between the early observations and the mean of the ProMP for the first portion of the trajectory:

$$\hat{k} = \arg\min_{k \in [1:K]} \left[\frac{1}{n_o} \sum_{t=1}^{n_o} |\Xi_t - \Phi_{\hat{\alpha}_k t} \mu_{\boldsymbol{\omega}_k}| \right]. \quad (15)$$

In equation (15), for each ProMP $k \in [1 : K]$, we compute the average distance between the observed early-trajectory Ξ_t and the mean trajectory of the ProMP $\Phi_{\hat{\alpha}_k t} \mu_{\omega_k}$, with $t = [1 : n_o]$. The most likely ProMP $\hat{k}$ is selected by computing the minimum distance (arg min). Other possible methods for estimating the most likely ProMPs could be inspired by those presented in the previous section for estimating the time modulation, i.e., maximum likelihood or learned models.

Once identified the $\hat{k}$-th most likely ProMP, we update its posterior distribution to take into account the initial portion of the observed trajectory, using equation (8):

$$\begin{cases} \hat{\mu}_{\boldsymbol{\omega}_{\hat{k}}} & = \mu_{\boldsymbol{\omega}_{\hat{k}}} + K\Big(\Xi^{o} - \Phi_{\hat{\alpha}_{\hat{k}}[1:n_o]}\mu_{\boldsymbol{\omega}_{\hat{k}}}\Big) \\ \hat{\Sigma}_{\boldsymbol{\omega}_{\hat{k}}} & = \Sigma_{\boldsymbol{\omega}_{\hat{k}}} - K\Big(\Phi_{\hat{\alpha}_{\hat{k}}[1:n_o]}\Sigma_{\boldsymbol{\omega}_{\hat{k}}}\Big) \\ K & = \Sigma_{\boldsymbol{\omega}_{\hat{k}}}\,\Phi^{\top}_{\hat{\alpha}_{\hat{k}}[1:n_o]}\Big(\Sigma_{\xi^{o}} + \Phi_{\hat{\alpha}_{\hat{k}}[1:n_o]}\Sigma_{\boldsymbol{\omega}_{\hat{k}}}\Phi^{\top}_{\hat{\alpha}_{\hat{k}}[1:n_o]}\Big)^{-1} \end{cases} \tag{16}$$

with $\hat{\alpha}_{\hat{k}}[1:n_o] = \hat{\alpha}_{\hat{k}}\,t$ (in matrix form), with $t \in [1:n_o]$.

Finally, the inferred trajectory is given by the following equation:

$$\forall t \in [1:\hat{t}_f], \hat{\xi}(t) = \Phi_t\,\hat{\mu}_{\boldsymbol{\omega}_{\hat{k}}}$$

with the expected duration of the trajectory $\hat{t}_f = \hat{\alpha}_k \bar{s}$. The robot is now able to finish the movement executing the most likely "future" trajectory $\hat{\Xi} = [\hat{\xi}_{n_o+1} \ldots \hat{\xi}_{t_f}]^{\mathrm{T}}$.

4. SOFTWARE OVERVIEW

In this section, we introduce our open-source software with an overview of its architecture. This software is composed of two main modules, represented in **Figure 5**.

While the robot is learning the Probabilistic Movement Primitives (ProMPs) associated with the different tasks, the robot is controlled by its user. The user's guidance can be either manual for the real iCub, or through a haptic device for the simulated robot.

A Matlab module allows replaying movement primitives or finishing a movement that has been initiated by its user. By using this module, the robot can learn distributions over trajectories, replay movement primitives (using the mean of the distribution), recognize the ProMP that best matches a current trajectory, and infer the future evolution (until the end target) of this trajectory.

A C++ module forwards to the robot the control that comes either from the user or from the Matlab module. Then, the robot is able to finish a movement initiated by its user (directly or through a haptic device) in an autonomous way, as shown in **Figure 1**.

We present the C++ module in Section 6.2 and the theoretical explanation of the Matlab module algorithms in Section 3. A guide to run this last module is first presented in Section 5 for a simple example, and in Section 6 for our application, where a simulated robot learns many measured information of the movements. Finally, we present results on the real iCub application in Section 7.

Our software is available through the GPL license, and publicly available at:

https://github.com/inria-larsen/icubLearningTrajectories.

Tutorial, readme, and videos can be found in that repository. First, the readme file describes how to launch simple demonstrations of the software. Videos present these demonstrations to simplify the understanding. In the next sections, we detail the operation of the demo program for a first case of 1-DOF primitive, followed by the presentation of the specific applications on the iCub (first simulated and then real).

5. SOFTWARE EXAMPLE: LEARNING A 1-DOF PRIMITIVE

In this section, we present the use of the software to learn ProMPs in a simple case of 1-DOF primitive. This example only uses the *MatlabProgram* folder, composed of:

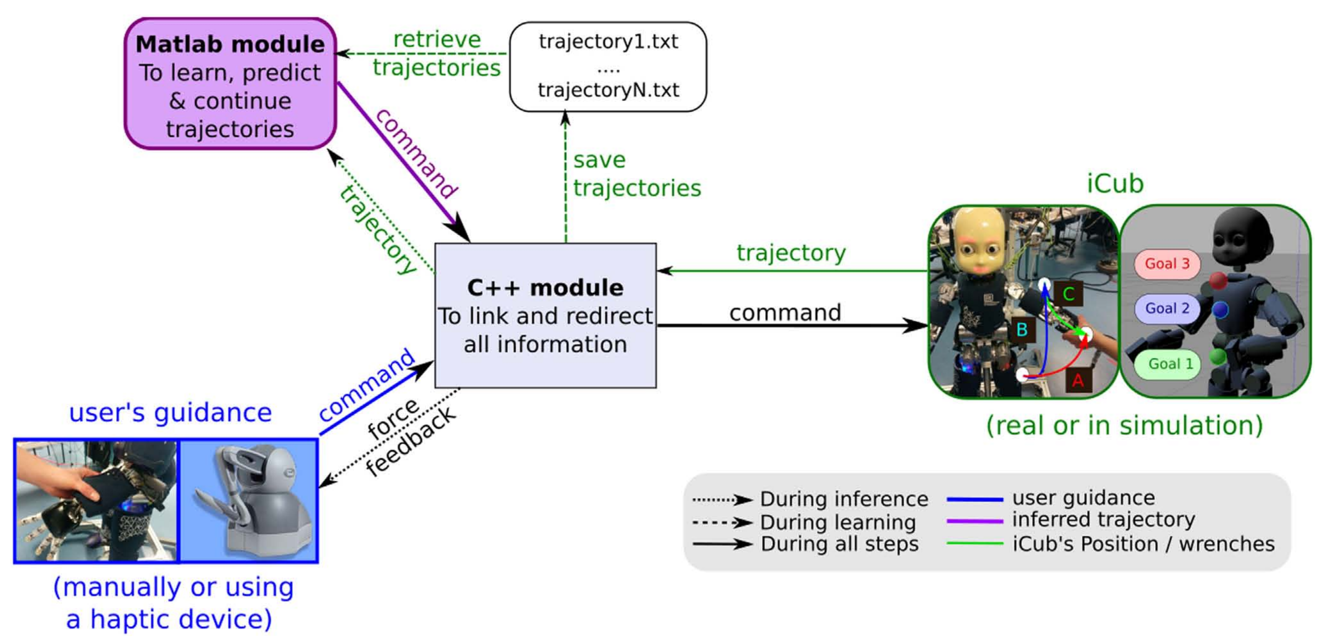

FIGURE 5 | Software architecture and data flows. The robot's control is done either by the user's guidance (manually or through a haptic device) represented in blue, or by the Matlab module, in purple. The C++ module handles the control source to command the robot, as represented in black. Moreover, this module forwards information that comes from the iCub.

- A sub-folder called "Data," where there are trajectory sets used to learn movement primitives. These trajectories are stored in text files with the following information:
  - **input parameters**: # input$_1$ # input$_2$ [...]
  - **input parameters with time step**: # timeStep # input$_1$ # input$_2$ [...]
  - ***recordTrajectories.cpp*** *program recording*: See Section 6.3 for more information.
- A sub-folder called "used_functions." It contains all the functions used to retrieve trajectories, compute ProMPs, infer trajectories, and plot results. Normally, using this toolbox does not require understanding these functions. The first lines of these functions give an explanation of their functioning and precise what are the input(s) and output(s) parameters.
- Matlab scripts called "demo_*.m." They are simple examples of how to use this toolbox.

The script `demo_plot1DOF.m`, can be used to compute a ProMP and to continue an initiated movement. The ProMP is computed from a dataset stored in a ".mat" file, called *traj1_1DOF.mat*. In this script, variables are first defined to make the script specific to the current dataset:

Variable assignation	Commentary
DataPath = 'Datatraj1_1DOF.mat';	Can be either ".mat" or ".txt". In the current demo, you can also write DataPath = 'Data/traj1' if you want to use the text files of this dataset.
typeRecover = '.mat'	Or .txt, it depends on your choice of data file.
inputName = {'z[m]'};	Label of your input(s). Here *z* represents the *z*-axis Cartesian coordinate.
s_ref = 100;	Number of samples used as reference to rescale all the trajectories to the same duration.
nbInput = 1;	Dimension of the generic vector containing the state of the trajectory.
M = 5;	Number of radial basis functions per input.
expNoise = 0.00001;	Expected trajectory noise.
percentData = 20;	Percent of observed data during the inference.

The variables include the following:

- `DataPath` is the path to the recorded data. If the data are stored in text files, this variable contains the folder name where text files are stored. These text files are called "record*X*.txt," with $X \in [0: n-1]$ if there are n trajectories. One folder is used to learn one ProMP. If the data are already loaded from a ".mat" file, write the whole path with the extension. The data in ".mat" match with the output of the Matlab function `loadTrajectory`.
- `nbInput` $= D$ is the dimension of the input vector ξ_t.
- `expNoise` $= \Sigma^o_\xi$ is the expected noise of the initiated trajectory. The smaller this variable is, the stronger the modification of the ProMP distribution will be, given new observations.

We will now explain more in detail the script. To recover data recorded in a ".txt" file, we call the function:

```
t{1} = loadTrajectory(PATH, nameT, varargin)
```

Its input parameters specify the path of the recorded data, the label of the trajectory. Other information can be added by using the `varargin` variable (for more detail, check the header of the function with the help comments). The output is an object that contains all the information about the demonstrated trajectories. It is composed of `nbTraj`, the number of trajectory; `realTime`, the simulation time; and `y` (and `yMat`), the vector (and matrix) trajectory set. Thus, `t{1}.y{i}` contains the *i*-th trajectory.

The Matlab function `drawRecoverData(t{1}, inputName,'namFig', nFig, varargin)` plots in a Matlab figure (numbered `nFig`) the dataset of loaded trajectories. An example is shown in **Figure 2**, on the left. Incidentally, the different duration of the trajectories is visible: on average, it is 1.17 ± 0.42 s.

To split the entire dataset of demonstrated trajectories `t{1}` into a training dataset (used for learning the ProMPs) and a test dataset (used for the inference), call the function

```
[train, test] = partitionTrajectory(t{1},
partitionType, percentData, s_ref)
```

where if `partitionType` $= 1$, only one trajectory is in the test set and the others are placed in the training set, and if `partitionType` > 1 it corresponds to the percentage of trajectories that will be included in the training set.

The ProMP can be computed from the training set by using the function:

```
promp = computeDistribution(train, M, s_ref, c, h)
```

The output variable `promp` is an object that contains all the ProMP information. The first three input parameters have been presented before: `train` is the training set, `M` is the number of RBFs, and `s_ref` is the number of samples used to rescale all the trajectories. The last two input parameters `c` and `h` shape the RBFs of the ProMP model: `c` $\in \mathbb{R}^M$ is the center of the Gaussians and `h` $\in \mathbb{R}$ their variance.

To visualize this ProMP, as shown in **Figure 2**, call the function:

```
drawDistribution(promp, inputName, s_ref)
```

For debugging purposes and to understand how to tune the ProMPs' parameters, it is interesting to plot the overlay of the basis functions in time. Choosing an appropriate number of basis functions is important, as too few may be insufficient to approximate the trajectories under consideration, and too many could result in overfitting problems. To plot the basis functions, simply call:

```
drawBasisFunction(promp.PHI, M)
```

where `promp.PHI` is a set of RBFs evaluated in the normalized time range $t \in [1 : \bar{s}]$.

Figure S1 in Supplementary Material shows at the top the basis functions before normalization, and at the bottom the ProMP modeled from these basis functions. From left to right, we change the number of basis functions. When there are not enough basis functions (left), the model is not able to correctly represent the shape of the trajectories. In the middle, the trajectories are well represented by the five basis functions. With more basis functions (right), the variance of the distribution decreases because the model is more accurate. However, arbitrarily increasing the number of basis functions is not a good idea, as it may not improve the accuracy of the model and worse it may cause overfitting.

Once the ProMP is learned, the robot can reproduce the movement primitive using the mean of the distribution. Moreover, it can now recognize a movement that has been initiated in this distribution and predict how to finish it. To do so, given the early n_o observations of a movement, the robot updates the prior distribution to match the early observed data points: through conditioning, it finds the posterior distribution, which can be used by the robot to execute the movement on its own.

The first step in predicting the evolution of the trajectory is to infer the duration of this trajectory, which is encoded by the time modulation parameter $\hat{\alpha}$. The computation of this inference, which was detailed in Section 3.4, can be done by using the function:

```
[expAlpha, type, x] = inferenceAlpha(promp,
test{1}, M, s_ref, c, h, test{1}.nbData,...
expNoise, typeReco)
```

where `typeReco` is the type of criteria used to find the expected time modulation ("MO," "DI," or "ML" for model, distance or maximum likelihood methods); `expAlpha` $= \hat{\alpha}$ is the expected time modulation; `type` is the index of the ProMP from which `expAlpha` has been computed, which we note in this article as k. To predict the evolution of the trajectory, we use equation (8) from Section 3.3. In Matlab, this is done by the function:

```
infTraj = inference(promp, test{1}, M, s_ref, c,
h,... test{1}.nbData, expNoise, expAlpha).
```

where `test{1}.nbData` has been computed during the `partitionTrajectory` step. This variable is the number of observations n_o, representing the percentage of observed data (`percentData`) of the test trajectory (i.e., to be inferred) that the robot observes. `infTraj`= $\hat{\Xi}$ is the inferred trajectory. Finally, to draw the inferred trajectory, we can call the function `drawInference(promp, inputName, infTraj, test1, s_ref)`.

It can be interesting to plot the quality of the predicted trajectories as a function of the number of observations, as done in **Figure 3**.

Note that when we have observed a larger portion of the trajectory, the prediction of the remaining portion is more accurate.

Now we want to measure the quality of the prediction. Let $\Xi^* = [\xi^o(1), \ldots, \xi^o(n_o), \xi^*(n_o+1), \ldots, \xi^*(t_f^*)]$ be the real trajectory expected by the user. To measure the quality of the prediction, we can use:

- The likelihood of having the Ξ^* trajectory given the updated distribution $p(\hat{\boldsymbol{\omega}})$.
- The distance between the Ξ^* trajectory and the $\hat{\Xi}$ inferred trajectory.

However, according to the type of recognition `typeReco` used to estimate the time modulation parameter α from the early observations, a visible mismatch between the predicted trajectory and the real one can be visible even when a lot of observations are used. This is due to the error of the expectation of this time modulation parameter. In Section 3.4, we present the different methods used to predict the trajectory duration. These methods select the most likely $\hat{\alpha}$ according to different criteria: distance; maximum likelihood; model of the α variable[11]; and average of the observed α during learning.

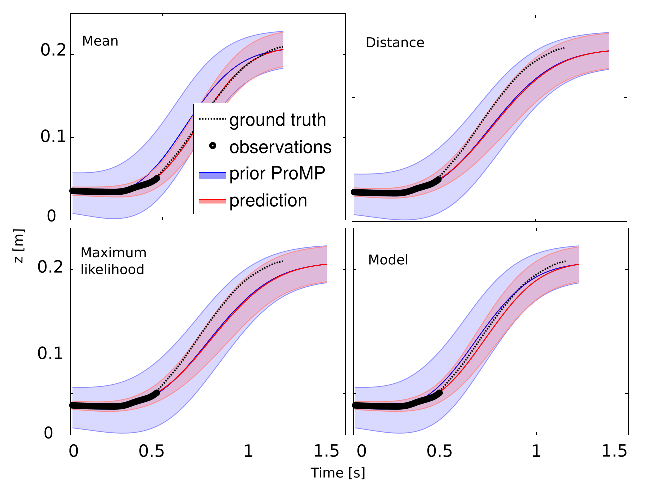

FIGURE 6 | The prediction of the future trajectory given $n_o = 40\%$ of early observations from the learned ProMP computed for the test dataset (**Figure 2**). The plots show the predicted trajectory, using different criteria to estimate the best phases of the trajectory: using the average time modulation (equation (11)); using the distance criteria (equation (13)); using the maximum log-likelihood (equation (12)); or using a model of time modulation according to the time variation (equation (14)).

TABLE 2 | Information about trajectories' duration.

	Traj. samples	$\alpha = \frac{\bar{s}}{\text{Iterations}}, \bar{s} = 100$	Duration [s]
Min	83	1.2048	0.83
Max	115	0.8696	1.15
Mean	100	1	0.99
SD	9	11.1111	0.09

Figure 6 shows the different trajectories predicted after $n_o = 40\%$ of the length of the desired trajectory is observed, according to the method used to estimate the time modulation parameter.

On this simple test, where the variation time is little as shown in **Table 2**, the best result is accomplished by the average of time modulation parameter of the trajectories used during the learning step. In more complicated cases, when the time modulation varies, the other methods will be preferable as seen in Section 3.5.

6. APPLICATION ON THE SIMULATED iCub: LEARNING THREE PRIMITIVES

In this application, the robot learns multiple ProMPs and is able to predict the future trajectory of a movement initiated by the user, assuming the movement belongs to one of the learned primitives.

[11] In this model, we assume that we can find the time modulation parameter according to the global variation of the position during the n_o first observed data.

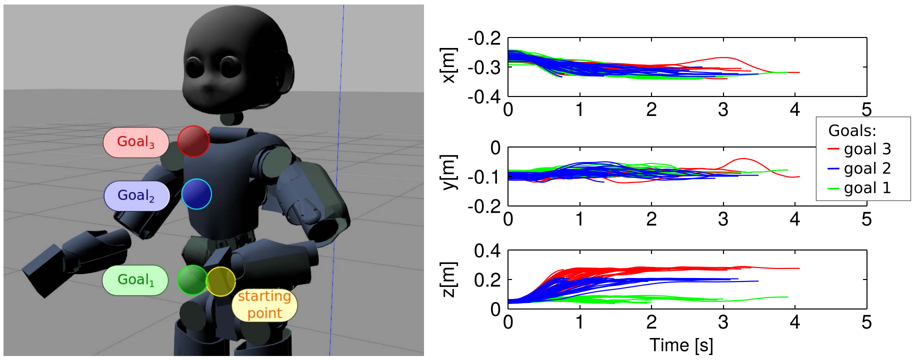

FIGURE 7 | Left: the three colored targets that the robot must reach from the starting point; the corresponding trajectories are used to learn three primitives representing three skills. Right: the Cartesian position information of the demonstrated trajectories for the three reaching tasks.

Based on this prediction, it can also complete the movement once it has recognized the appropriate ProMP.

We simplify the three actions/tasks by reaching three different targets, represented by three colored balls in the reachable workspace of the iCub. The example is performed with the simulated iCub in Gazebo. **Figure 7** shows the three targets, placed at different heights in front of the robot.

In Section 6.1, we formulate the intention recognition problem for the iCub: the problem is to learn the ProMP from trajectories consisting of Cartesian positions in 3D[12] and the 6D wrench information measured by the robot during the movement. In Section 6.2, we describe the simulated setup of iCub in Gazebo, then in Section 6.3, we explain how trajectories are recorded, including force information, when we use the simulated robot.

6.1. Predicting Intended Trajectories by Using ProMPs

The model is based on Section 3, but here we want to learn more information during movements. We record this information in a multivariate parameter vector:

$$\forall t, \xi_t = \begin{bmatrix} X_t \\ F_t \end{bmatrix} \in \mathbb{R}^9,$$

where $X_t \in \mathbb{R}^3$ is the Cartesian position of the robot's end effector and $F_t \in \mathbb{R}^6$ the external forces and moments. In particular, F_t contains the user's contact forces and moments. Let us call $\dim(\xi_t) = D$, the dimension of this parameter vector.

The corresponding ProMP model is as follows:

$$\xi_t = \begin{bmatrix} X_t \\ F_t \end{bmatrix} = \Phi_{\alpha t}\boldsymbol{\omega} + \epsilon_t,$$

where $\boldsymbol{\omega} \in \mathbb{R}^{D \cdot M}$ is the time-independent parameter vector, $\epsilon_t = \begin{bmatrix} \epsilon_{X_t} \\ \epsilon_{F_t} \end{bmatrix} \in \mathbb{R}^D$ is the zero-mean Gaussian i.i.d. observation noise, and $\Phi_{\alpha t} \in \mathbb{R}^{D \times D \cdot M}$ a matrix of Radial Basis Functions (RBFs) evaluated at time αt.

Since we are in the multidimensional case, this $\Phi_{\alpha t}$ block diagonal matrix is defined as follows:

$$\Phi_{\alpha t} = BlockdiagonalMatrix(\phi_1, \ldots, \phi_D) \in R^{D \times D \cdot M}.$$

It is a diagonal matrix of D Radial Basis Functions (RBFs), where each RBF represents one dimension of the ξ_t vector and it is composed of M Gaussians, spread over same number of samples $\bar{s}$.

6.1.1. Learning Motion Primitives

During the learning step of each movement primitive $k \in [1:3]$, the robot observes different trajectories $S_k = \{\Xi_1, \ldots, \Xi_n\}_k$, as presented in Section 6.3.

For each trajectory $\Xi_{i[1:t_{fi}]} = [\xi_{i(1)}, \ldots, \xi_{i(t_{fi})}]^{\mathrm{T}}$, it computes the optimal $\boldsymbol{\omega}_{ki}$ parameter vector that best approximates the trajectory.

We saw in Section 3.5 how these computations are done. In our software, we use matrix computation instead of t_{fi} iterative ones done for each observation t (as in equation (3)). Thus, we have the following:

$$\boldsymbol{\omega}_{ki} = \left(\Phi^{\top}_{\alpha[1:t_{fi}]}\Phi_{\alpha[1:t_{fi}]}\right)^{-1}\Phi^{\top}_{\alpha[1:t_{fi}]} * \Xi_{i[1:t_{fi}]} \tag{17}$$

with $\Phi_{\alpha[1:t_{fi}]} = [\Phi_{\alpha 1}, \Phi_{\alpha 2} \ldots, \Phi_{\alpha t_{fi}}]^{\mathrm{T}}$.

6.1.2. Prediction of the Trajectory Evolution from Initial Observations

After having learned the three ProMPs, the robot is able to finish an initiated movement on its own. In Sections 3.3–3.5, we explained how to compute the future intended trajectory given the early observations.

In this example, we add specificities about the parameters to learn.

Let $\Xi^o = \begin{bmatrix} X^o \\ F^o \end{bmatrix} = [\Xi_1 \ldots \Xi_{n_o}]^{\mathrm{T}}$ be the early observations of the trajectory.

First, we only consider the partial observations: $X^o = [X_1 \ldots X_{n_o}]^{\mathrm{T}}$. Indeed, we assume the recognition of a trajectory is done with Cartesian position information only, because the

[12] Note that in that particular example we do not use the orientation because we want the robot's hand to keep the same orientation during the movement. But in principle, it is possible to learn trajectories consisting of the 6D/7D Cartesian position and orientation of the hand, to make the robot change also the orientation of the hand during the task.

same movement can be done and recognized with different force profiles than the learned ones.

From this partial observation X^o, the robot recognizes the current ProMP $\hat{k} \in [1:3]$, as seen in Section 3.5. It also computes an expectation of the time modulation $\hat{t}_f$, as seen in Section 3.4. Using the expected value of the time modulation, it approximates the trajectory speed and its total time duration.

Second, we use the total observation Ξ^o to update the ProMP, as seen in Section 3.3. This computation is based on equation (18), but here again, we use the following matrix computation:

$$\begin{cases} \hat{\mu}_{\boldsymbol{\omega}_k} &= \mu_{\boldsymbol{\omega}_k} + K\left(\Xi^o - \Phi_{\alpha[1:n_o]}\mu_{\boldsymbol{\omega}_k}\right) \\ \hat{\Sigma}_{\boldsymbol{\omega}_k} &= \Sigma_{\boldsymbol{\omega}_k} - K\left(\Phi_{\alpha[1:n_o]}\Sigma_{\boldsymbol{\omega}_k}\right) \\ K &= \Sigma_{\boldsymbol{\omega}_k}\Phi^{\mathrm{T}}_{\alpha[1:n_o]}\left(\Sigma_{\xi^o} + \Phi_{\alpha[1:n_o]}\Sigma_{\boldsymbol{\omega}_k}\Phi^{\mathrm{T}}_{\alpha[1:n_o]}\right)^{-1}. \end{cases}$$

From this posterior distribution, we retrieve the inferred $\hat{\Xi} = \{\hat{\xi}_1, ..., \hat{\xi}_{\hat{t}_f}\}$ trajectory, with:

$$\forall t \in [1:\hat{t}_f], \hat{\xi}_t = \begin{bmatrix} \hat{X}_t \\ \hat{F}_t \end{bmatrix} = \Phi_{\alpha t}\hat{\mu}_{\boldsymbol{\omega}_k}.$$

Note that the inferred wrenches $\hat{F}_t$, here, correspond to the simulated wrenches in Gazebo. In this example, there is little use for them in simulation; the interest for predicting also wrenches will be clearer in Section 7, with the example on the real robot.

6.2. Setup for Simulated iCub

For this application, we created a prototype in Gazebo, where the robot must reach three different targets with the help of a human. To interact physically with the robot simulated in Gazebo, we used the Geomagic touch, a haptic device.

The setup consists of the following:

- the iCub simulation in Gazebo, complete with the dynamic information provided by *wholeBodyDynamicsTree* (https://github.com/robotology/codyco-modules/tree/master/src/modules/wholeBodyDynamicsTree) and the Cartesian information provided by *iKinCartesianController*;
- the Geomagic Touch, installed following the instructions in https://github.com/inria-larsen/icub-manual/wiki/Installation-with-the-Geomagic-Touch, which not only install the SDK and the drivers of the GeoMagic but also point to how to create the yarp drivers for the Geomagic;
- a C++ module (https://github.com/inria-larsen/icubLearningTrajectories/tree/master/CppProgram) that connects the output command from the Geomagic to the iCub in Gazebo and eventually enables recording the trajectories on a file. A tutorial is included in this software.

The interconnection among the different modules is represented in **Figure 5**, where the Matlab module is not used. The tip of the Geomagic is virtually attached to the end effector of the robot:

$$x_{geo} \longrightarrow x_{icub\_hand}.$$

When the operator moves the Geomagic, the position of the Geomagic tip x_{geo} is scaled (1:1 by default) in the iCub workspace as x_{icub_hand}, and the Cartesian controller is used to move the iCub hand around a "home" position, or default starting position:

$$x_{icub\_hand} = hapticDriverMapping(x_0 + x_{geo}),$$

where *hapticDriverMapping* is the transformation applied by the haptic device driver, which essentially maps the axis from the Geomagic reference frame to the iCub reference frame. By default, no force feedback is sent back to the operator in this application, as we want to emulate the zero-torque control mode of the real iCub, where the robot is ideally transparent and not opposing any resistance to the human guidance. A default orientation of the hand ("katana" orientation) is set.

6.3. Data Acquisition

The dark button of the Geomagic is used to start and stop the recording of the trajectories. The operator must click and hold the button during the whole movement and release the button at the end. The trajectory is saved on a file called *recordX.txt* for the X-th trajectory. The structure of this file is:

```
1 #time #xgeo #ygeo #zgeo #fx #fy #fz #mx #my #mz #x_icub_hand
  #y_icub_hand #z_icub_hand
2 5.96046e−06  −0.0510954  −0.0127809  −0.0522504  0.284382
  −0.0659538  −0.0239582  −0.0162418  −0.0290078  −0.0607215
  −0.248905 −0.0872191 0.0477496$
```

A video showing the iCub's arm moved by a user through the haptic device in Gazebo is available in Section 8 (tutorial video). The graph in **Figure 7** represents some trajectories recorded with the Geomagic, corresponding to lifting the left arm of the iCub.

Demonstrated trajectories and their corresponding forces can be recorded directly from the robot, by accessing the Cartesian interface and the *wholeBodyDynamicsTree* module.[13]

In our project on Github, we provide the acquired dataset with the trajectories for the interested reader who wishes to test the code with these trajectories. Two datasets are available at https://github.com/inria-larsen/icubLearningTrajectories/tree/master/MatlabProgram/Data/: the first dataset called "heights" is composed of three goal-directed reaching tasks, where the targets vary in height; the second dataset called "FLT" is composed of trajectories recorded on the real robot, whose arms move forward, to the left and to the top.

A Matlab script that learns ProMPs with such kinds of datasets is available in the toolbox, called `demo_plotProMPs.m`. It contains all the following steps.

To load the first "heights" dataset with the three trajectories, write:

```
1 t{1}=loadTrajectory('Data/heights/bottom', 'bottom', 'refNb',
  s_bar, 'nbInput', nbInput, ... 'Specific', 'FromGeom');
2 t{2}=loadTrajectory('Data/heights/top', 'top', 'refNb',
  s_bar, 'nbInput', nbInput, ... 'Specific', 'FromGeom');
3 t{3}=loadTrajectory('Data/heights/middle', 'forward',
  'refNb', s_bar, 'nbInput', nbInput, ... 'Specific',
  'FromGeom');
```

[13] In our example, we do not use the simulated wrench information as it is very noisy. However, we provide the code and show how to retrieve it and use it, in case the readers should not have access to the real iCub.

Figure 7 shows the three sets of demonstrated trajectories. In the used dataset called "heights," we have recorded 40 trajectories per movement primitive.

6.4. Learning the ProMPs

We need to first learn the ProMPs associated with the three observed movements. First, we partition the collected dataset into a training set and test dataset for the inference. One random trajectory for the inference is used:

```
[train{i}, test{i}] = partitionTrajectory(t{i}, 1, percentData,
s_bar);
```

The second input parameter specifies that we select only one trajectory, randomly selected, to test the ProMP.

Now, we compute the three ProMPs with:

```
promp{1} = computeDistribution(train{1}, M, s_bar, c, h);
promp{2} = computeDistribution(train{2}, M, s_bar, c, h);
promp{3} = computeDistribution(train{3}, M, s_bar, c, h)
```

We set the following parameters:

- `s_bar = 100`: reference number of samples, which we note in this article as $\bar{s}$.
- `nbInput(1) = 3; nbInput(2) = 6`: dimension of the generic vector containing the state of the trajectories. It is composed of 3D Cartesian position and 6D forces and wrench information.[14]
- `M(1) = 5;  M(2) = 5`: number of basis functions for each `nbInput` dimension.
- `c = 1/M;h = 1/(M*M)`: RBF parameters (see equation (2)).
- `expNoise = 0.00001`: the expected data noise. We assume this noise to be very low, since this is a simulation.
- `percentData = 40`: this variable specifies the percentage of the trajectory that the robot will be observed, before inferring the end.

These parameters can be changed at the beginning of the Matlab script.

Figure 8 shows the three ProMPs of the reaching movements toward the three targets. To highlight the most useful dimension, we only plot the z-axis Cartesian position. However, each dimension is plotted by the Matlab script with:

```
drawRecoverData(t{1}, inputName, 'Specolor', 'b', 'namFig', 1);
drawRecoverData(t{1}, inputName, 'Interval', [4 7 5 8 6 9],
'Specolor', 'b', 'namFig', 2);
drawRecoverData(t{2}, inputName, 'Specolor', 'r', 'namFig', 1);
drawRecoverData(t{2}, inputName, 'Interval', [4 7 5 8 6 9],
'Specolor', 'r', 'namFig', 2);
drawRecoverData(t{3}, inputName, 'Specolor', 'g', 'namFig', 1);
drawRecoverData(t{3}, inputName, 'Interval', [4 7 5 8 6 9],
'Specolor', 'g', 'namFig', 2);
```

[14] Note that in our example wrenches are separated from the Cartesian position, because they are not used to recognize the index of the current ProMP during the inference.

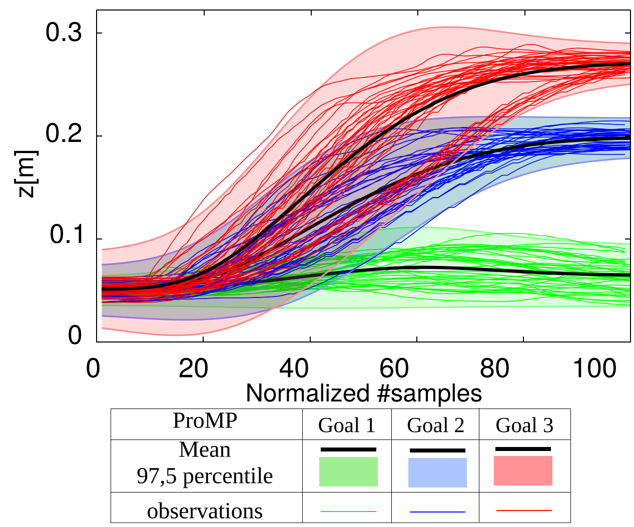

FIGURE 8 | The Cartesian position in the z-axis of the three ProMPs corresponding to reaching three targets. There are 39 trajectory demonstrations per each ProMPs with M = 5 basis functions, $c = \frac{1}{M}, h = \frac{1}{M^2}$ and $\bar{s} = 100$.

6.5. Predicting the Desired Movement

Now that we have learned the different ProMPs, we can predict the end of a trajectory according to the early observation n_o. This number is computed from the variable `percentData` written at the beginning of the trajectory by: $n_o = \lfloor \frac{percentData}{100} * t_{fi} \rfloor$, where i is the index of the test trajectory.

To prepare the prediction, the model the time modulation of each trajectory is computed with:

```
w = computeAlpha(test.nbData, t, nbInput);
promp1.w_alpha = w1;
promp2.w_alpha = w2;
promp3.w_alpha = w3;
```

This model relies on the global variation of Cartesian position during the first n_o observations. The model's computations are explained in Section 3.4.

Now, to estimate the time modulation of the trajectory, call the function:

```
[alphaTraj, type, x] = inferenceAlpha(promp, test{1}, M, s_bar,
c, h, test{1}.nbData, expNoise, 'MO');
```

where `alphaTraj` contains the estimated time modulation $\hat{\alpha}$ and `type` gives the index of the recognized ProMP. The last parameter `x` is used for debugging purposes.

Using this estimation of the time modulation, the end of the trajectory is inferred with:

```
infTraj = inference(promp, test{1}, M, s_bar, c, h,
test{1}.nbData, expNoise, alphaTraj);
```

As shown in the previous example, the quality of the prediction of the future trajectory depends on the accuracy of the time modulation estimation. This estimation is computed by calling the function:

```
%Using the model:
[alphaTraj, type, x] = inferenceAlpha(promp, test{1}, M, s_bar,
  c, h, test{1}.nbData, expNoise, 'MO');
%Using the distance criteria:
[alphaTraj, type, x] = inferenceAlpha(promp, test{1}, M, s_bar,
  c, h, test{1}.nbData, expNoise, 'DI');
%Using the Maximum likelihood criteria:
[alphaTraj, type, x] = inferenceAlpha(promp, test{1}, M, s_bar,
  c, h, test{1}.nbData, expNoise, 'ML');
%Using the mean of observed temporal modulation during learning:
alphaTraj = (promp{1}.mu_alpha + promp{2}.mu_alpha + promp{3}.
  mu_alpha)/3.0;
```

6.6. Predicting the Time Modulation

In Section 3.4, we presented four main methods for estimating the time modulation parameter, discussing why this is crucial for a better estimation of the trajectory. Here, we compare the methods on the three goals experiment. We recorded 40 trajectories for each movement primitive, for a total of 120 trajectories. After having computed the corresponding ProMPs, we tested the inference by providing early observations of a trajectory that the robot must finish. For that purpose, it recognizes the correct ProMP among the three precedently learned (see Section 3.5) and then it estimates the time modulation parameter $\hat{\alpha}$. **Figure 9** represents the average error of the $\hat{\alpha}$ during inference for 10 trials according to the number of observations (from 30 to 90% of observed data) and according to the used method. These methods are the ones we have just presented before that we called mean (equation (11)), maximum likelihood (equation (12)), minimum distance (equation (13)) or model (equation (14)). Each time, the tested trajectory is chosen randomly from the data set of observed trajectories (of course, the test trajectory does not belong to the training set, so it was not used in the learning step). The method that takes the average of α observed during learning is taken as comparison (in black). We can see that other methods are more accurate. The *maximum likelihood* is increasingly more accurate, as expected. The fourth method (*model*) that models the α according to the global variation of the trajectory's positions during the early observations is the best performing when the portion of observed trajectory is small (e.g., 30–50%). Since it is our interest to predict the future trajectory as early as possible, we adopted the *model* method for our experiments.

7. APPLICATION ON THE REAL iCub

In this section, we present and discuss two experiments with the real robot iCub.

In the first, we take inspiration from the experiment of the previous Section 6, where the "tasks" are exemplified by simple point-to-point trajectories demonstrated by a human tutor. In this experiment, we explore how to use wrench information and use known demonstrations as ground truth, to evaluate the quality of our prediction.

In the second experiment, we set up a more realistic collaborative scenario, inspired by collaborative object sorting. In such applications, the robot is used to lift an object (heavy, or dangerous, or that the human cannot manipulate, as for some chemicals or food), the human inspects the object and then decides if it is accepted or rejected. Depending on this decision, the object goes on a tray or bin in front of the robot, or on a bin located on the robot side. Dropping the objects in two cases must be done in a different way. Realizing this application with iCub is not easy, as iCub cannot lift heavy objects and has a limited workspace. Therefore, we simplify the experiment with small objects and two bins. The human simply starts the robots movement with physical guidance, and then the robot finishes the movement on its own. In this experiment the predicted trajectories are validated on-the-fly by the human operator.

In a more complex collaborative scenario, tasks could be elementary tasks such as pointing, grasping, reaching, and manipulating tools (the type of task here is not important, as long as it can be represented by a trajectory).

7.1. Three Simple Actions with Wrench Information

Task trajectories, in this example, have both position and wrench information. In general, it is a good idea to represent collaborative motion primitives in terms of both position and wrenches, as this representation enables using them in the context of physical interaction. Contrarily to the simulated experiment, here the inferred wrenches $\hat{F}_t$ correspond to the wrenches the robot should perceive if the partner was manually guiding the robot to perform the entire movement: indeed, these wrenches are computed from the demonstrations used to learn the primitive. The predicted wrenches can be used in different ways, depending on the application. For example, if the partner breaks the contact with the robot, the perceived wrenches will be different. If the robot is not equipped with tactile or contact sensors, this information can be used by the robot to "perceive" the contact breaking and interpret it, for example, as the sign that the human wants the robot to continue the task on its own. Another use for the demonstrated wrenches is for detecting abnormal forces while the robot is moving: this use can have different applications, from adapting the motion to new environment to automatically detecting new demonstrations. Here, they are simply used to detect when the partner breaks the contact with the robot, and the latter must continue the movement on its own.

In the following, we present how to realize the experiment for predicting the user intention with the real iCub, using our software. The robot must learn three task trajectories represented in **Figure 10**. In red, the first trajectory goes from an initial position in front of the robot to its left (task A). In green, the second trajectory goes from the same initial position to the top (task C). In blue, the last trajectory goes from the top position to the position on the left (task B).

To provide the demonstrations for the tasks, the human tutor used three visual targets shown on the iCub_GUI, a basic module of the iCub code that provides a real-time synthetic and augmented view of the robot status, with arrows for the external forces and colored objects for the targets. One difficulty for novice

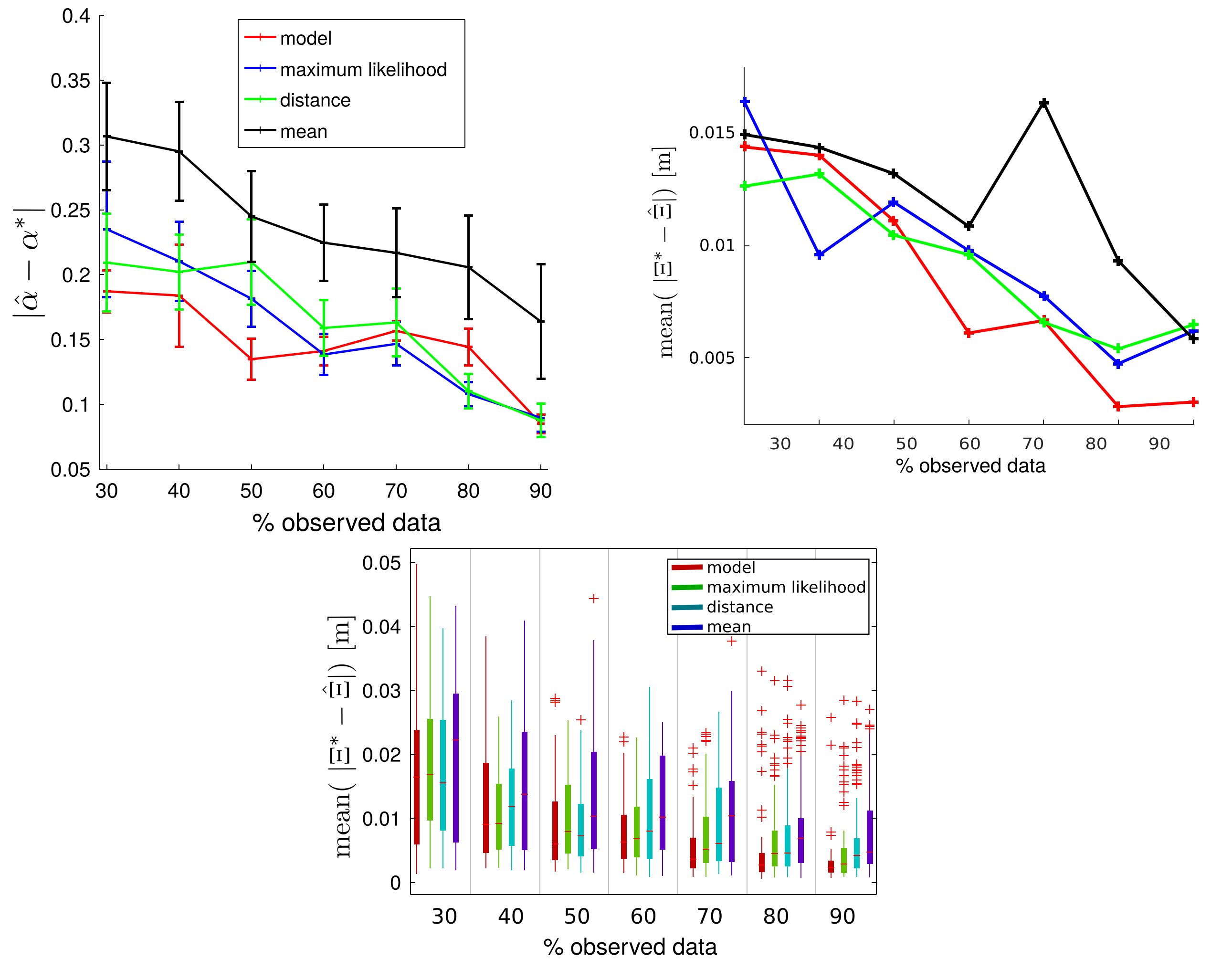

FIGURE 9 | (Top left) Error of α estimation; (top right and bottom) error of trajectory prediction according to the number of known data and the method used. We executed 10 different trials for each case.

users of iCub is to be able to drive the robot's arm making it perform desired complex 3D trajectories (Ivaldi et al., 2017), but after some practice in moving the robot's arm the operator recorded all the demonstrations. We want to highlight that having variations in the starting or ending points of the trajectories is not at all a problem, since the ProMPs are able to deal with this variability.

We will see that by using the ProMPs method and by learning the end-effector Cartesian position, the robot will be able to learn distributions over trajectories, recognize when a movement belongs to one of these distributions, and infer the end of the movement.

In this experiment, the robot received 10 demonstrated trajectories per movement primitive, all provided by the same user. We recorded the Cartesian end-effector position and the wrenches of the robot's left arm. Data are retrieved using the function `used_functions/retrieveRealDataWithoutOrientation.m`. The output parameters of this function are three objects (one per ProMP) that contain all the required information to learn the ProMPs.

In this function, the wrench information are filtered using a Matlab function called `envelope.m`[15]: for each trajectory `traj` and its subMatrix $M = F([1{:}\, t])$:

```
[envHigh, envLow] = envelope(traj.M);
traj.M = (envHigh + envLow)/2;
```

These three objects are saved in `'Data/realIcub.mat'`. A Matlab script called `demo_plotProMPsIcub.m` recovers these data, using the function `load('Data/realIcub.mat')`. This script follows the same organization as the ones we previously explained in Sections 5 and 6. By launching this script, the recovered data are plotted first.

Then, the ProMPs are computed and plotted, as presented in **Figure 11**. In this figure, the distributions are visibly overlaid:

- during the whole trajectories duration for the wrench information;

[15] Information about this function can be found here: https://fr.mathworks.com/help/signal/ref/envelope.html?requestedDomain=www.mathworks.com.

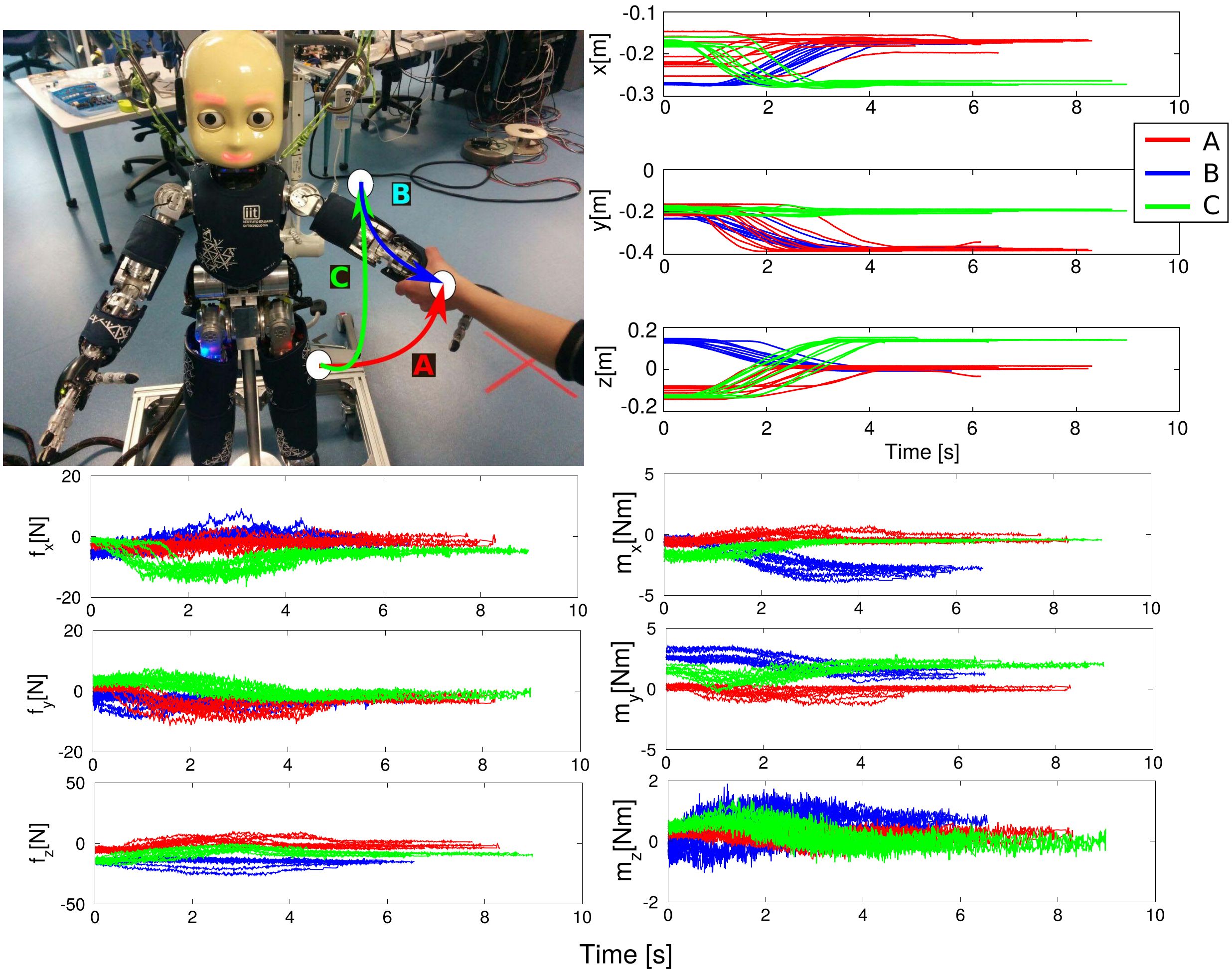

FIGURE 10 | Top left: the iCub and the visualization of the three targets in its workspace, defining the three tasks A–B–C. Top right: Cartesian position information of the demonstrated trajectories for the three tasks. Bottom left and right: wrench (force and moment) information of the demonstrated trajectories.

- during the 40% first samples of the trajectories for the Cartesian position information.

After this learning step, the user chooses which ProMP to test. Using a variable that represents the percentage of observed data to be used for the inference, the script computes the number of early observations n_o[16] that will be measured by the robot. Using this number, the robot models the time modulation parameter α[17] of each ProMP, as explained in Section 3.4. Using this model, the time modulation of the test trajectory is estimated, and the corresponding ProMP is identified.

Then, the inference of the trajectory's target is performed. **Figure 12** represents the inference of the three tested trajectories when wrench information is not used by the robot to infer the trajectory. To realize this figure, with the comparison between the predicted trajectory and the ground truth, we applied our algorithm offline. In fact, it is not possible at time t to have the ground truth of the trajectory intended by the human from $t+1$ to t_f: even if we would tell to the human in advance the goal that he/she must reach for, the trajectory to reach that goal could vary. So, for the purpose of these figures and comparisons with the ground truth, we show here the offline evaluation: we select one demonstrated task trajectory from the test set (not the training set used to learn the ProMP) as ground truth, and imagine that this is the intended trajectory. In **Figure 12**, the ground truth is shown in black, whereas the portion of this trajectory that is fed to the inference, and that corresponds to the "early observations," is represented with bigger black circles. We can see that the inference of the Cartesian position is correct, although we can see an error of about 1 s of the estimated duration time for the last trial. Also, the wrench inference is not accurate. We can assume that it is: because the robot infers the trajectory using only position information without wrench information, or because the wrenches' variation is not correlated to the position variation. To improve this result, we can make the inference using wrench in addition to Cartesian position information, as shown in **Figure 13**. We can see in this figure that the estimation of the trajectory's duration is accurate. The disadvantage is that the inference of the Cartesian position

[16] n_o is not the same for each trajectory test, because it depends on the total duration of the trajectory to be inferred.

[17] Since the model uses the n_o parameter, its computation cannot be performed before this step.

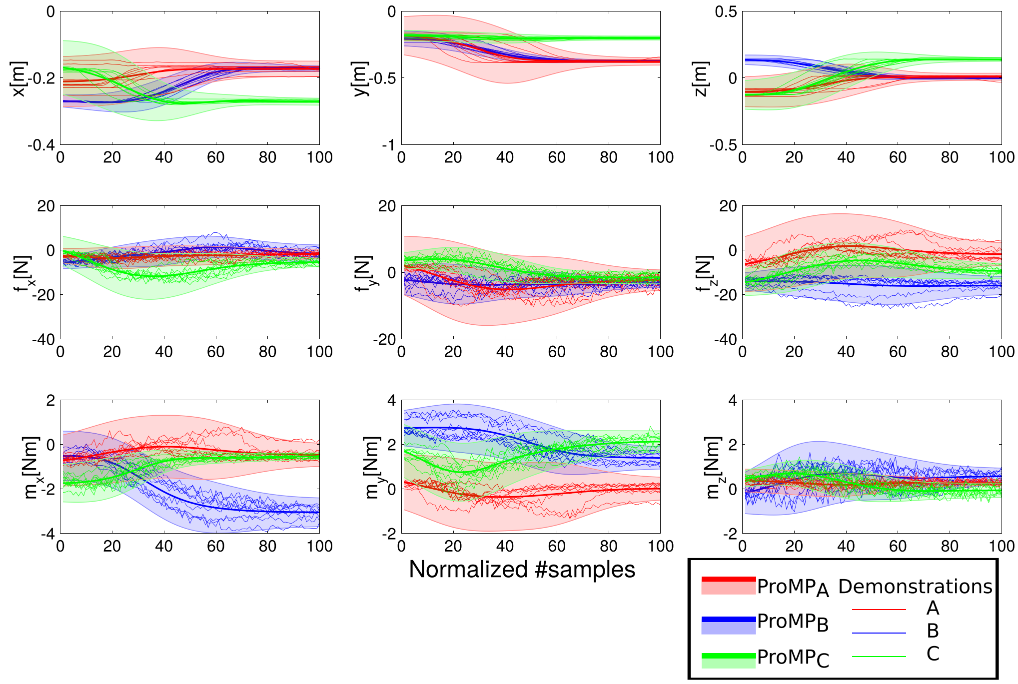

FIGURE 11 | The ProMPs learned by the robot from the demonstrations of **Figure 10**.

is less accurate because the posterior distribution computation makes a trade-off between fitting Cartesian position and wrench early observations. Moreover, to allow a correct inference using wrench information, the noise expectation must be increased to consider forces.[18]

To confirm these results, we analyzed the trajectory inference and α estimation considering different percentages of each trajectory as observed data (30–90%). For each percentage, we performed 20 tests, with and without force information.

In **Figure 14**, each box-plot represents errors for 20 tests. On the top, the error criterion is the average distance between the inferred trajectory and the real one. We can see that the inference of Cartesian end-effector trajectory is more accurate without wrench information. On the bottom, the error criterion is the distance between the estimated α and the real one. We can see that using wrench information, the estimation of the α is more accurate. Thus, these two graphs confirm what we assumed from **Figures 12** and **13**.

Median, mean, and variance of the prediction errors, computed with the normalized root-mean-square error (NRMSE), are reported in Table S1 in Supplementary Material. The prediction error for the time modulation is a scalar: $|\alpha_{\text{prediction}} - \alpha_{\text{real}}|$. The prediction error for the trajectory is computed by the NRMSE of $|\Xi_{\text{prediction}} - \Xi_{\text{real}}|$.

In future upgrades for this application, we will probably use the wrench information only to estimate the time modulation parameter α, to have both the best inference of the intended trajectory and the best estimation of the time modulation parameter to combine the benefits of inference with and without wrench information.

Table S1 in Supplementary Material also reports the average time for computing the prediction of both time modulation and posterior distribution. The computation was performed in Matlab, on a single core laptop (no parallelization). While the computation time for the case "without wrenches" is fine for real-time application, using the wrench information delays the prediction and represents a limit for real-time applications if fast decisions have to taken by the robot. Computation time will be improved in the future works, with the implementation of the prediction in an iterative way.

7.2. Collaborative Object Sorting

We realized another experiment with iCub, where the robot has to sort some objects in different bins (see Figure S2 in Supplementary Material). We have two main primitives: one for a bin located on the left of the robot, and one for the bin to the front. Dropping the object is done at different heights, with a different gesture that also has a different orientation of the hand. For this reason, the ProMP model consists of the Cartesian position of the hand $X_t = [x_t, y_t, z_t] \in R^3$ and its orientation $A_t \in R^4$, expressed as a quaternion:

$$\xi_t = \begin{bmatrix} X_t \\ A_t \end{bmatrix} = \Phi_{\alpha t}\boldsymbol{\omega} + \epsilon_t.$$

As in the previous experiment, we first teach the robot the primitives by kinesthetic teaching, with a dozen of demonstrations. Then we start the robot movement: the human operator physically grabs the robot's arm and start the movement toward one of the bins. The robot's skin is used twice. First, to detect the contact

[18] In future versions, we will include the possibility to have different noise models for the observations, e.g., we will have $\Sigma^o_\Xi = \begin{bmatrix} \Sigma_X & 0 \\ 0 & \Sigma_F \end{bmatrix}$. We will therefore set a bigger covariance for the wrench information than for the position information.

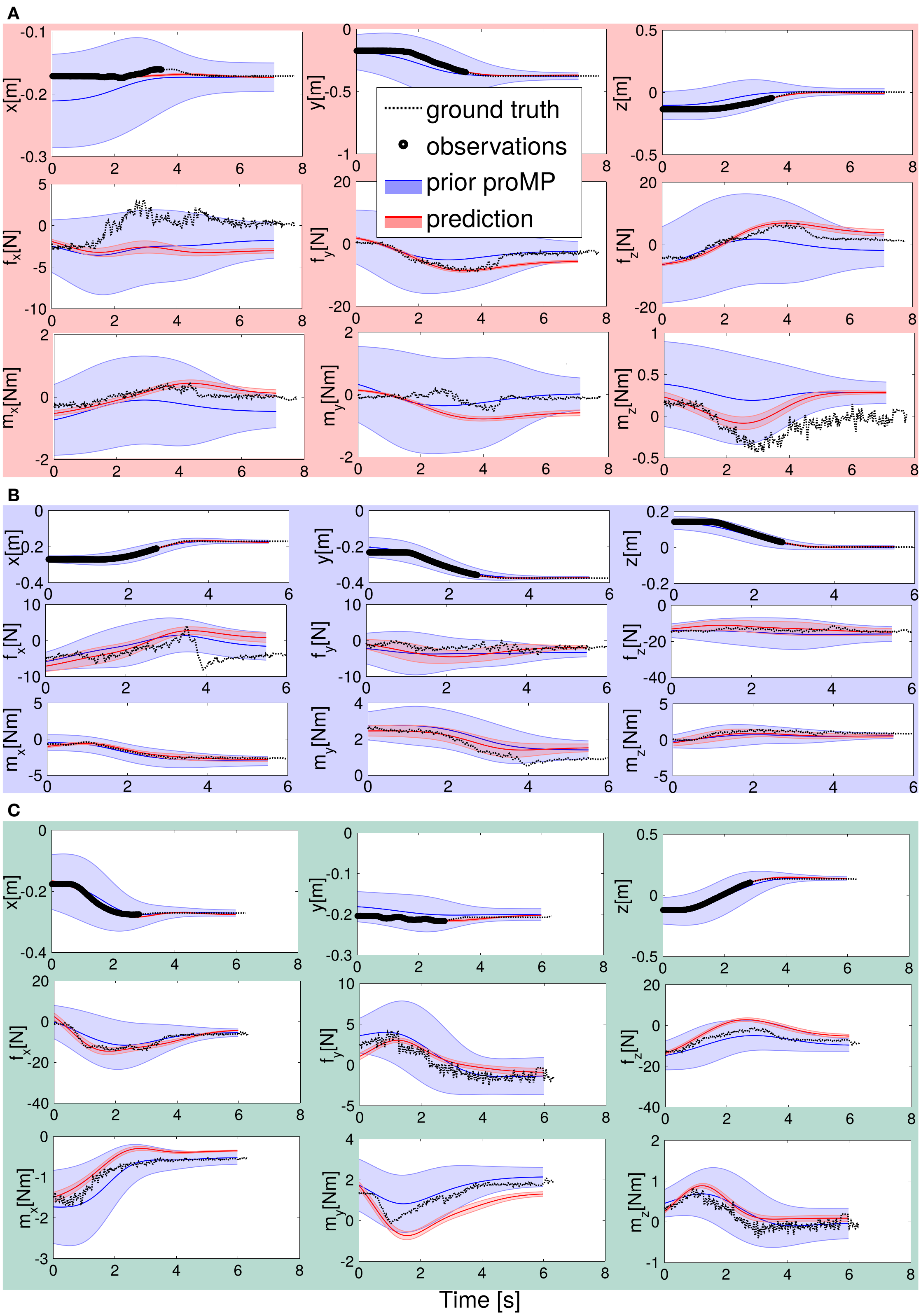

FIGURE 12 | The prediction of the future trajectory from the learned ProMPs computed from the position information for the 3-targets dataset on the real iCub (**Figure 11**) after 40% of observations.

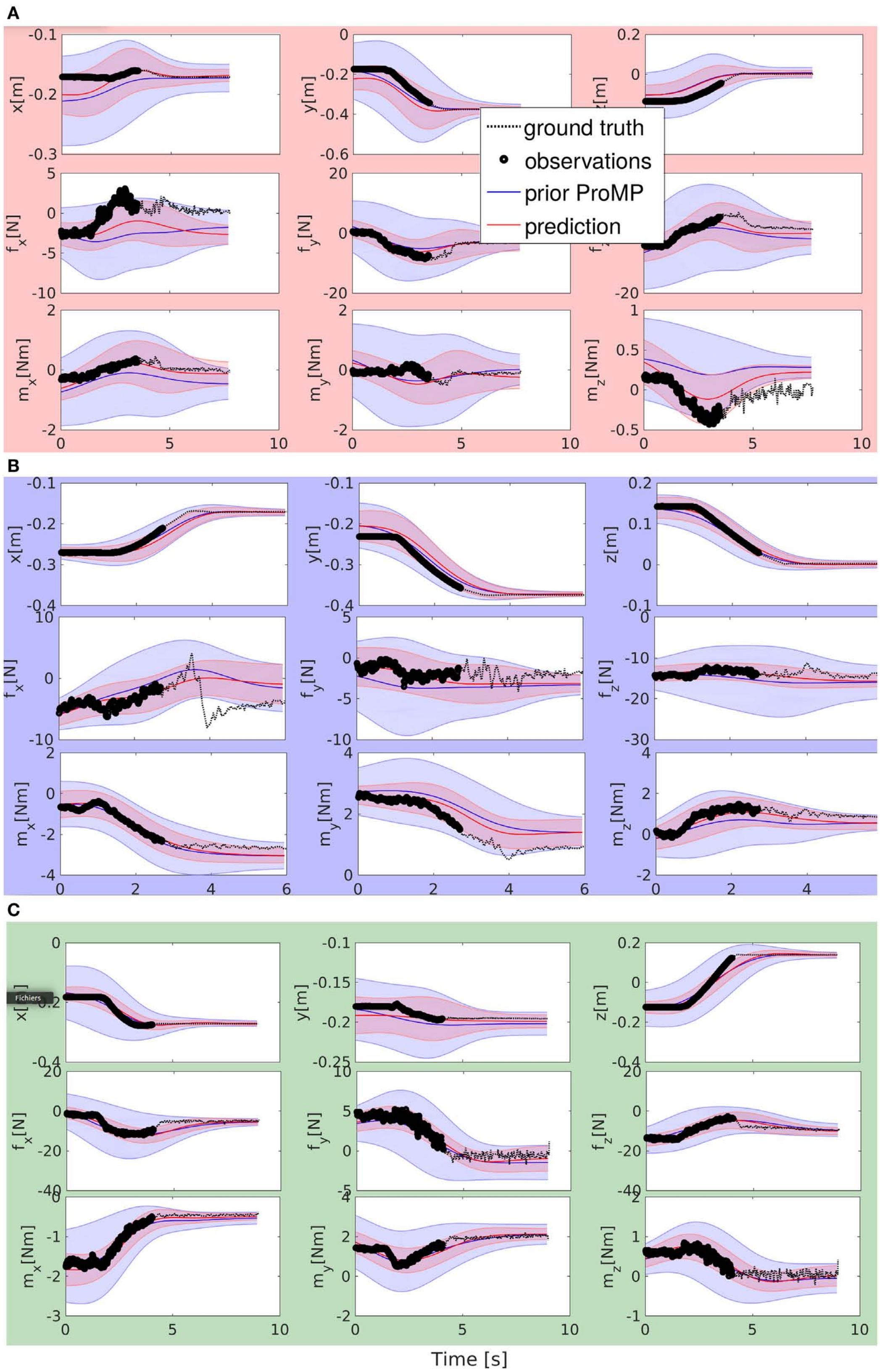

FIGURE 13 | The prediction of the future trajectory from the learned ProMPs computed from the position and wrench information for the 3-targets dataset on the real iCub (**Figure 11**) after 40% of observations.

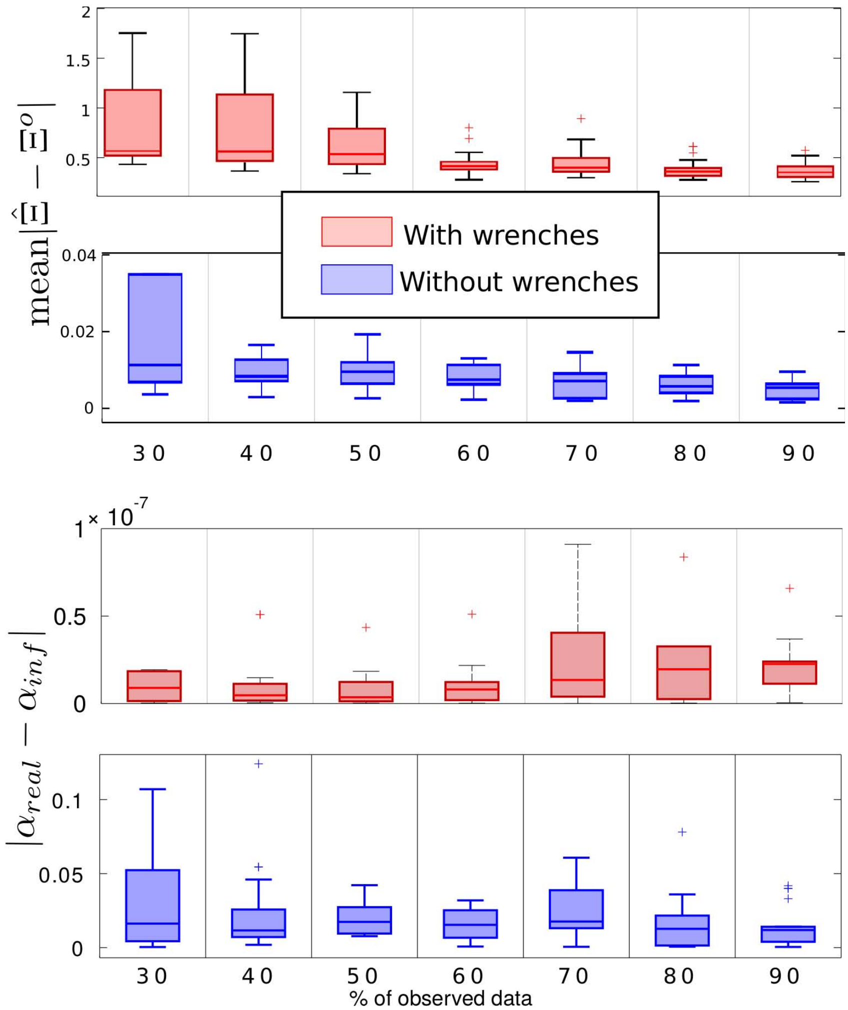

FIGURE 14 | Trajectory prediction error (top) and time modulation estimation error (bottom) of the future trajectory with and without wrench information, for the 3-targets dataset on the real iCub (**Figure 11**) with respect to the number of observed data points.

when the human grabs the arm, which marks the beginning of the observations. Second, when the human breaks the contact with the arm, which marks the end of the observations. Using the first portion of the observed movement, the robot recognizes the current task that is being executed, predicts the future movement that is intended by the human, and then executes it on its own. In the video (see link in Section 8), we artificially introduced a pause to let the operator "validate" the predicted trajectory, using a visual feedback on the iCubGui. Figure S3 in Supplementary Material shows one of the predictions made by the robot after the human releases the arm. Of course in this case, we do not have a "ground truth" for the predicted trajectory, only a validation of the predicted trajectory by the operator.

8. VIDEOS

We recorded several videos that complement the tutorials. The videos are presented in the github repository of our software: https://github.com/inria-larsen/icubLearningTrajectories/tree/master/Videos.

9. DISCUSSION

While we believe that our proposed method is principled and has several advantages for predicting intention in human–robot interaction, there are numerous improvements that can be done. Some will be object of our future works.

9.1. Improving the Estimation of the Time Modulation

Our experiments showed that estimating the time modulation parameter α, determining the duration of the trajectory, greatly improves the prediction of the trajectory in terms of difference with the human intended trajectory (i.e., our ground truth). We proposed four simple methods in Section 3.4, and in the iCub experiment, we showed that the method that maps the time modulation and the variation of the trajectory in the first n_o observations provides a good estimate of the time modulation α for our specific application. However, it is an *ad hoc* model that cannot be generalized to all possible cases. Overall, the estimation of the time modulation (or phase) can be improved. For example, Maeda et al. (2016) used Dynamic Time Warping, while Ewerton et al. (2015) proposed to improve the estimation by having local estimations of the speed in the execution of the trajectory, to comply with cases where the velocity of task trajectory may not be constant throughout the task execution. In the future, we plan to explore more solutions and integrate them into our software.

9.2. Improving Prediction

Another point that needs further investigation and improvement is how to improve the prediction of the trajectories exploiting different information. In our experiment with iCub, we improved the estimation of the time modulation using position and wrench information; however, we observed that the noisy wrench information does not help in improving the prediction of the position trajectory. One improvement is to certainly exploit more information from the demonstrated trajectories, such as estimating the different noise of every trajectory component and exploiting this information to improve the prediction. Another possible improvement would consist in using contextual information about the task trajectories. Finally, it would be interesting to try to identify automatically the characteristic such as velocity profiles or accelerations, which are renown to play a key role in attributing intentions to human movements. For example, in goal-directed tasks such as reaching, the arm velocity profile, and the hand configuration are cues that helps us detect intentions. Extracting these cues automatically, leveraging the estimation of the time modulation, would probably improve the prediction of the future trajectory. This is a research topic on its own, outside the scope of this article, with strong links to human motor control.

9.3. Continuous Prediction

In Section 3.5, we described how to compute the prediction of the future trajectory after recognizing the current task. However, we did not explore what happens if the task recognition is wrong: this may happen, if there are two or more task with a similar trajectory at the beginning (e.g., moving the object from the same initial point toward one of four possible targets), or simply because there were not enough observed points. So what happens if our task recognition is wrong? How to re-decide on a previously identified task? And how should the robot decide if its current prediction is finally correct (in statistical terms)? While implementing a continuous recognition and prediction is easy with our framework (one has simply to do the estimation at each time step), providing a generic answer to these question may not be straightforward. Re-deciding about the current task implies also changing the prediction of the future trajectory. If the decision does not come with a confidence level greater than a desired value, then the robot could face a stall: if asked to continue the movement but unsure about the future trajectory, should it continue or stop? The choice may be application dependent. We will address these issues and the continuous prediction in future works.

9.4. Improving Computational Time

Finally, we plan to improve the computational time for the inference and the portability of our software by porting the entire framework in C++.

9.5. Learning Tasks with Objects

In many collaborative scenarios, such as object carrying and cooperative assembly, the physical interaction between the human and the robot is mediated by objects. In these cases, if specific manipulations must be done on the objects, our method still applies, but not only on the robot. It must be adapted to the new "augmented system" consisting of robot and object. Typically, we could image a trajectory for some frame or variable or point of interest for the object and learn the corresponding task. Since ProMPs support multiplication and sequencing of primitives, we could exploit the properties of the ProMPs to learn the joint distribution of the robot task trajectories and the object task trajectories.

10. CONCLUSION

In this article, we propose a method for predicting the intention of a user physically interacting with the iCub in a collaborative task. We formalize the intention prediction as predicting the target and "future" intended trajectory from early observations of the task trajectory, modeled by Probabilistic Movement Primitives (ProMPs). We use ProMPs because they capture the variability of the task, in the form of a distribution of trajectories coming from several demonstrations of the task. From the information provided by the ProMP, we are able to compute the future trajectory by conditioning the ProMP to match the early observed data points. Additional features of our method are the estimation of the duration of the intended movement, the recognition of the current task among the many known in advance, and multimodal prediction.

Section 3 described the theoretical framework, whereas Sections 4–7 presented the open-source software that provides the implementation of the proposed method. The software is available on `github`, and tutorials and videos are provided.

We used three examples of increasing complexity to show how to use our method for predicting the intention of the human in collaborative tasks, exploiting the different features. We presented experiments with both the real and the simulated iCub. In our experiments, the robot learns a set of motion primitives corresponding to different tasks, from several demonstrations provided by a user. The resulting ProMPs are the prior information that is later used to make inferences about human intention. When the human starts a new collaborative task, the robot uses the early observations to infer which task the human is executing and predicts the trajectory that the human intends to execute. When the human releases the robot, the predicted trajectory is used by the robot to continue executing the task on its own.

In Section 9, we discussed some current issues and challenges for improving the proposed method and make it applicable to a wider repertoire of collaborative human–robot scenarios. In our future works, our priority would be in accelerating the time for computing the inference and finding a principled way to do continuous estimation, by letting the robot re-decide continuously about the current task and future trajectory.

AUTHOR CONTRIBUTIONS

Designed study: OD, AP, FC, and SI. Wrote software: OD, ME, AP, and SI. Wrote paper: OD, AP, ME, FC, JP, and SI.

ACKNOWLEDGMENTS

The authors wish to thank the IIT researchers of the CoDyCo project for their support with iCub, Ugo Pattacini and Olivier Rochel for their help with the software for the Geomagic, and Iñaki Fernández Pérez for all his relevant feedback.

FUNDING

This study was partially funded by the European projects CoDyCo (no. 600716 ICT211.2.1) and AnDy (no. 731540 H2020-ICT-2016-1) and the French CPER project SCIARAT.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at http://journal.frontiersin.org/article/10.3389/frobt.2017.00045/full#supplementary-material.

REFERENCES

Alami, R., Clodic, A., Montreuil, V., Sisbot, E. A., and Chatila, R. (2006). "Toward human-aware robot task planning," in *AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before*, Beijing, 39–46.

Amor, H. B., Neumann, G., Kamthe, S., Kroemer, O., and Peters, J. (2014). "Interaction primitives for human-robot cooperation tasks," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on* (Hong Kong: IEEE), 2831–2837.

Baraglia, J., Cakmak, M., Nagai, Y., Rao, R., and Asada, M. (2016). "Initiative in robot assistance during collaborative task execution," in *Human-Robot Interaction (HRI), 2016 11th ACM/IEEE International Conference on* (Christchurch: IEEE), 67–74.

Billard, A., and Mataric, M. J. (2001). Learning human arm movements by imitation: evaluation of a biologically inspired connectionist architecture. *Rob. Auton. Syst.* 37, 145–160. doi:10.1016/S0921-8890(01)00155-5

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

Busch, B., Grizou, J., Lopes, M., and Stulp, F. (2017). Learning legible motion from human–robot interactions. *Int. J. Soc. Robot.* 1–15. doi:10.1007/s12369-017-0400-4

Buxton, H. (2003). Learning and understanding dynamic scene activity: a review. *Image Vision Comput.* 21, 125–136. doi:10.1016/S0262-8856(02)00127-0

Calinon, S. (2015). *pbdlib-matlab*. Available at: https://gitlab.idiap.ch/rli/pbdlib-matlab/

Calinon, S. (2016). A tutorial on task-parameterized movement learning and retrieval. *Intell. Serv. Robot.* 9, 1–29. doi:10.1007/s11370-015-0187-9

Calinon, S., Bruno, D., and Caldwell, D. G. (2014). "A task-parameterized probabilistic model with minimal intervention control," in *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, 3339–3344.

Calinon, S., Li, Z., Alizadeh, T., Tsagarakis, N. G., and Caldwell, D. G. (2012a). *Statistical Dynamical Systems for Skills Acquisition in Humanoids*. Available at: http://www.calinon.ch/showPubli.php?publi=3031

Calinon, S., Li, Z., Alizadeh, T., Tsagarakis, N. G., and Caldwell, D. G. (2012b). "Statistical dynamical systems for skills acquisition in humanoids," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, Osaka, Japan, 323–329.

Carlson, T., and Demiris, Y. (2008). "Human-wheelchair collaboration through prediction of intention and adaptive assistance," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on* (Pasadena: IEEE), 3926–3931.

Caron, S., and Kheddar, A. (2016). "Multi-contact walking pattern generation based on model preview control of 3d com accelerations," in *Humanoid Robots, 2016 IEEE-RAS International Conference on*, Cancun.

Csibra, G., and Gergely, G. (2007). 'Obsessed with goals': functions and mechanisms of teleological interpretation of actions in humans. *Acta Psychol.* 124, 60–78. doi:10.1016/j.actpsy.2006.09.007

Demiris, Y. (2007). Prediction of intent in robotics and multi-agent systems. *Cogn. Process.* 8, 151–158. doi:10.1007/s10339-007-0168-9

Dermy, O. (2017). *icubLearningTrajectories*. Available at: https://github.com/inria-larsen/icubLearningTrajectories

DeWolf, T. (2013). *pydmps*. Available at: https://github.com/studywolf/pydmps

Dragan, A., and Srinivasa, S. (2013). "Generating legible motion," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany.

Dragan, A., and Srinivasa, S. (2014). Integrating human observer inferences into robot motion planning. *Auton. Robots* 37, 351–368. doi:10.1007/s10514-014-9408-x

Dumora, J., Geffard, F., Bidard, C., Aspragathos, N. A., and Fraisse, P. (2013). "Robot assistance selection for large object manipulation with a human," in *IEEE International Conference on Systems, Man, and Cybernetics*, Manchester, 1828–1833.

Evrard, P., Gribovskaya, E., Calinon, S., Billard, A., and Kheddar, A. (2009). "Teaching physical collaborative tasks: object-lifting case study with a humanoid," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on* (Paris: IEEE), 399–404.

Ewerton, M. (2016). *Learning Motor Skills from Partially Observed Movements Executed at Different Speeds*. Available at: https://github.com/studywolf/pydmps

Ewerton, M., Neumann, G., Lioutikov, R., Ben Amor, H., Peters, J., and Maeda, G. (2015). "Learning multiple collaborative tasks with a mixture of interaction primitives," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on* (Seattle: IEEE), 1535–1542.

Ferrer, G., and Sanfeliu, A. (2014). Bayesian human motion intentionality prediction in urban environments. *Pattern Recognit. Lett.* 44, 134–140. doi:10.1016/j.patrec.2013.08.013

Fine, S., Singer, Y., and Tishby, N. (1998). The hierarchical hidden Markov model: analysis and applications. *Mach. Learn.* 32, 41–62. doi:10.1023/A:1007469218079

Fitts, P. M. (1992). The information capacity of the human motor system in controlling the amplitude of movement. *J. Exp. Psychol. Gen.* 121, 262. doi:10.1037/0096-3445.121.3.262

Fumagalli, M., Ivaldi, S., Randazzo, M., Natale, L., Metta, G., Sandini, G., et al. (2012). Force feedback exploiting tactile and proximal force/torque sensing. *Auton. Robots* 33, 381–398. doi:10.1007/s10514-012-9291-2

Gribovskaya, E., Kheddar, A., and Billard, A. (2011). "Motion learning and adaptive impedance for robot control during physical interaction with humans," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (Shanghai: IEEE), 4326–4332.

Hersch, M., Guenter, F., Calinon, S., and Billard. (2008). *Dynamical System Modulation for Robot Adaptive Learning via Kinesthetic Demonstrations*. Available at: http://lasa.epfl.ch/sourcecode/counter.php?ID=11&index=1

Hoffman, G. (2010). "Anticipation in human-robot interaction," in *AAAI Spring Symposium: It's All in the Timing*, Palo Alto, CA.

Huang, S. H., Held, D., Abbeel, P., and Dragan, A. D. (2017). Enabling robots to communicate their objectives. *CoRR*. abs/1702.03465. Available at: http://arxiv.org/abs/1702.03465

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Comput.* 25, 328–373. doi:10.1162/NECO_a_00393

Ivaldi, S., Anzalone, S., Rousseau, W., Sigaud, O., and Chetouani, M. (2014a). Robot initiative in a team learning task increases the rhythm of interaction but not the perceived engagement. *Front. Neurorobot.* 8:5. doi:10.3389/fnbot.2014.00005

Ivaldi, S., Nguyen, S. M., Lyubova, N., Droniou, A., Padois, V., Filliat, D., et al. (2014b). Object learning through active exploration. *IEEE Trans. Auton. Ment. Dev.* 6, 56–72. doi:10.1109/TAMD.2013.2280614

Ivaldi, S., Fumagalli, M., Nori, F., Baglietto, M., Metta, G., and Sandini, G. (2010). "Approximate optimal control for reaching and trajectory planning in a humanoid robot," in *Proc. of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS*, Taipei, Taiwan, 1290–1296.

Ivaldi, S., Fumagalli, M., Randazzo, M., Nori, F., Metta, G., and Sandini, G. (2011). "Computing robot internal/external wrenches by means of inertial, tactile and f/t sensors: theory and implementation on the icub," in *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on* (Bled: IEEE), 521–528.

Ivaldi, S., Lefort, S., Peters, J., Chetouani, M., Provasi, J., and Zibetti, E. (2017). Towards engagement models that consider individual factors in HRI: on the relation of extroversion and negative attitude towards robots to gaze and speech during a human-robot assembly task. *Int. J. Soc. Robot.* 9, 63–86. doi:10.1007/s12369-016-0357-8

Jamone, L., Ugur, E., Cangelosi, A., Fadiga, L., Bernardino, A., Piater, J., et al. (2017). Affordances in psychology, neuroscience and robotics: a survey. *IEEE Trans. Cognit. Dev. Syst.* 99, 1. doi:10.1109/TCDS.2016.2594134

Jarrassé, N., Paik, J., Pasqui, V., and Morel, G. (2008). "How can human motion prediction increase transparency?," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on* (Pasadena, CA: IEEE), 2134–2139.

Keogh, E. (2002). "Exact indexing of dynamic time warping," in *Proceedings of the 28th International Conference on Very Large Data Bases* (Hong Kong: VLDB Endowment), 406–417.

Khansari, M. (2011). *Dynamical Systems Approach to Learn Robot Motions*. Available at: https://bitbucket.org/khansari/seds

Khansari-Zadeh, S. M., and Billard, A. (2011). Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Trans. Robot.* 27, 943–957. doi:10.1109/TRO.2011.2159412

Khansari-Zadeh, S. M., and Billard, A. (2012). A dynamical system approach to realtime obstacle avoidance. *Auton. Robots* 32, 433–454. doi:10.1007/s10514-012-9287-y

Kim, J., Banks, C. J., and Shah, J. A. (2017). "Collaborative planning with encoding of users' high-level strategies," in *AAAI Conference on Artificial Intelligence (AAAI-17)*, San Francisco, CA.

Langolf, G. D., Chaffin, D. B., and Foulke, J. A. (1976). An investigation of fitts? Law using a wide range of movement amplitudes. *J. Mot. Behav.* 8, 113–128. doi:10.1080/00222895.1976.10735061

Lober, R. (2014). *Stochastic Machine Learning Toolbox*. Available at: https://github.com/rlober/smlt

Lober, R., Padois, V., and Sigaud, O. (2014). "Multiple task optimization using dynamical movement primitives for whole-body reactive control," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on* (Madrid: IEEE), 193–198.

Maeda, G., Ewerton, M., Lioutikov, R., Ben Amor, H., Peters, J., and Neumann, G. (2014). "Learning interaction for collaborative tasks with probabilistic movement primitives," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on* (Madrid: IEEE), 527–534.

Maeda, G. J., Neumann, G., Ewerton, M., Lioutikov, R., Kroemer, O., and Peters, J. (2016). "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks," in *Autonomous Robots*, 1–20.

Meier, F., and Schaal, S. (2016). A probabilistic representation for dynamic movement primitives. *CoRR*. abs/1612.05932. Available at: http://arxiv.org/abs/1612.05932

Micha, H., and Aude, B. (2008). Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Trans. Robot.* 24, 1463–1467. doi:10.1109/TRO.2008.2006703

Nguyen, N. T., Phung, D. Q., Venkatesh, S., and Bui, H. (2005). "Learning and detecting activities from movement trajectories using the hierarchical hidden Markov model," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, Vol. 2 (San Diego, CA: IEEE), 955–960.

Palinko, O., Sciutti, A., Patané, L., Rea, F., Nori, F., and Sandini, G. (2014). "Communicative lifting actions in human-humanoid interaction," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on* (Madrid: IEEE), 1116–1121.

Paraschos, A., Daniel, C., Peters, J. R., and Neumann, G. (2013a). "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems*, Stateline, NV, 2616–2624.

Paraschos, A., Neumann, G., and Peters, J. (2013b). "A probabilistic approach to robot trajectory generation," in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on* (Atlanta, GA: IEEE), 477–483.

Paraschos, A., Rueckert, E., Peters, J., and Neumann, G. (2015). "Model-free probabilistic movement primitives for physical interaction," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on* (Hamburg: IEEE), 2860–2866.

Pattacini, U., Nori, F., Natale, L., Metta, G., and Sandini, G. (2010). "An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on* (Taipei: IEEE), 1668–1674.

Peters, J., Lee, D. D., Kober, J., Nguyen-Tuong, D., Bagnell, J. A., and Schaal, S. (2016). "Robot learning," in *Springer Handbook of Robotics*, 357–398.

Ren, H., and Xu, G. (2002). "Human action recognition with primitive-based coupled-HMM," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, Vol. 2 (Quebec: IEEE), 494–498.

Rozo Castañeda, L., Calinon, S., Caldwell, D., Jimenez Schlegl, P., and Torras, C. (2013). "Learning collaborative impedance-based robot behaviors," in *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 1422–1428.

Sahin, E., Çakmak, M., Dogar, M. R., Ugur, E., and Üçoluk, G. (2007). To afford or not to afford: a new formalization of affordances toward affordance-based robot control. *Adapt. Behav.* 15, 447–472. doi:10.1177/1059712307084689

Sato, T., Nishida, Y., Ichikawa, J., Hatamura, Y., and Mizoguchi, H. (1994). "Active understanding of human intention by a robot through monitoring of human behavior," in *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, Vol. 1, Munich, 405–414.

Schaal, S. (2006). "Dynamic movement primitives: a framework for motor control in humans and humanoid robotics," in *Adaptive Motion of Animals and Machines* Springer, 261–280.

Sciutti, A., Bisio, A., Nori, F., Metta, G., Fadiga, L., and Sandini, G. (2013). Robots can be perceived as goal-oriented agents. *Interact. Stud.* 14, 329–350. doi:10.1075/is.14.3.02sci

Shah, J., Wiken, J., Williams, B., and Breazeal, C. (2011). "Improved human-robot team performance using chaski, a human-inspired plan execution system," in *Proceedings of the 6th International Conference on Human-Robot Interaction* (Lausanne: ACM), 29–36.

Silva, D. F., and Batista, G. E. (2016). "Speeding up all-pairwise dynamic time warping matrix calculation," in *Proceedings of the 2016 SIAM International Conference on Data Mining* (Miami, FL: SIAM), 837–845.

Soechting, J. (1984). Effect of target size on spatial and temporal characteristics of a pointing movement in man. *Exp. Brain Res.* 54, 121–132. doi:10.1007/BF00235824

Soh, H., and Demiris, Y. (2015). Learning assistance by demonstration: smart mobility with shared control and paired haptic controllers. *J. Hum. Robot Interact.* 4, 76–100. doi:10.5898/JHRI.4.3.Soh

Stulp, F. (2014). *DmpBbo – a c++ Library for Black-Box Optimization of Dynamical Movement Primitives*. Available at: https://github.com/stulp/dmpbbo

Stulp, F., Raiola, G., Hoarau, A., Ivaldi, S., and Sigaud, O. (2013). "Learning compact parameterized skills with a single regression," in *Proc. IEEE-RAS International Conference on Humanoid RObots – HUMANOIDS*, Atlanta, GA, 1–7.

Thill, S., and Ziemke, T. (2017). "The role of intention in human-robot interaction," in *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, HRI '17* (New York, NY, USA: ACM), 427–428.

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2005). "Gaussian process dynamical models," in *NIPS*, Vol. 18, Vancouver, 3.

Wang, Z., Deisenroth, M. P., Amor, H. B., Vogt, D., Schölkopf, B., and Peters, J. (2012). "Probabilistic modeling of human movements for intention inference," in *Robotics: Science and Systems* (Sydney, NSW: Citeseer).

Wang, Z., Mülling, K., Deisenroth, M. P., Amor, H. B., Vogt, D., Schölkopf, B., et al. (2013). Probabilistic movement modeling for intention inference in human-robot interaction. *Int. J. Robot. Res.* 32, 841–858. doi:10.1177/0278364913478447

Zube, A., Hofmann, J., and Frese, C. (2016). "Model predictive contact control for human-robot interaction," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, Munich, 1–7.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

APPENDIX

A. Detail of the Inference Formula

In this Appendix, we explain how to obtain the inference formulae used in our software. First, let us recall the Marginal and Conditional Gaussians laws.[19] Given a marginal Gaussian distribution for x and a Gaussian distribution for y given x in the following form:

$$p(x) = \mathcal{N}\left(x|\mu, \Delta^{-1}\right) p(y|x) = \mathcal{N}\left(Ax + b, L^{-1}\right), \tag{A1}$$

the marginal distribution of y and the conditional distribution of x given y are given by the following equations:

$$p(y) = \mathcal{N}\left(y|A\mu + b, L^{-1} + A\Delta^{-1}A^{\top}\right), \tag{A2}$$

$$p(x|y) = \mathcal{N}\left(x|\Sigma A^{\top} L(y - b) + \Delta\mu, \Sigma\right), \tag{A3}$$

where

$$\Sigma = (\Delta + A^{\mathrm{T}}LA)^{-1}.$$

We computed the parameter's marginal Gaussian distribution from the set of observed movements:

$$p(\boldsymbol{\omega}) \sim \mathcal{N}(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}), \tag{A4}$$

[19] From the book (Bishop, 2006).

From the model $\Xi_t = \Phi_{[1:t_f]}\boldsymbol{\omega} + \epsilon_{\Xi}$, we have the conditional Gaussian distribution for Ξ given $\boldsymbol{\omega}$:

$$p(\Xi|\boldsymbol{\omega}) = \mathcal{N}\left(\Xi|\Phi_{[1:t_f]}\boldsymbol{\omega}, \Sigma_{\Xi}\right). \tag{A5}$$

Then, using equation (A2) we have the following:

$$p(\Xi) = \mathcal{N}\left(\Xi|\Phi_{[1:t_f]}\mu_{\boldsymbol{\omega}}, \Sigma_{\Xi} + \Phi_{[1:t_f]}\Sigma_{\boldsymbol{\omega}}\Phi_{[1:t_f]}^{\top}\right). \tag{A6}$$

that is the prior distribution of the ProMP.

Let $\Xi^o = [\xi^o(1), \ldots, \xi^o(n_o)]$ be the first n_o observations of the trajectory to predict with the first n_o elements corresponding to the early observations.

Let $\hat{\Xi} = [\xi^o(1), \ldots, \xi^o(n_o), \hat{\xi}(n_o+1), \ldots, \hat{\xi}(t_{\hat{t}_f})]$ be the whole trajectory we have to predict. We can then compute the posterior distribution of the ProMP by using the conditional Gaussians equation (A3):

$$p(\boldsymbol{\omega}|\Xi^o) = \mathcal{N}(\boldsymbol{\omega}|\mu_{\boldsymbol{\omega}} + K(\Xi^o - \Phi_{[1:n_o]}\mu_{\boldsymbol{\omega}}), \Sigma_{\boldsymbol{\omega}} - K\Phi_{[1:n_o]}\Sigma_{\boldsymbol{\omega}}) \tag{A7}$$

$$\text{with } K = \Sigma_{\boldsymbol{\omega}}\Phi_{[1:n_o]}^{\top}\left(\Sigma_{\Xi} + \Phi_{[1:n_o]}\Sigma_{\boldsymbol{\omega}}\Phi_{[1:n_o]}^{\top}\right)^{-1}. \tag{A8}$$

Thus, we have the posterior distribution of the ProMP $p(\boldsymbol{\omega}|\Xi^o) = \mathcal{N}(\boldsymbol{\omega}|\hat{\boldsymbol{\mu}}_{\omega}, \hat{\boldsymbol{\Sigma}}_{\omega})$ with:

$$\begin{cases} \hat{\mu}_{\boldsymbol{\omega}} &= \mu_{\boldsymbol{\omega}} + K\left(\Xi^o - \Phi_{[1:n_o]}\mu_{\boldsymbol{\omega}}\right) \\ \hat{\Sigma}_{\boldsymbol{\omega}} &= \Sigma_{\boldsymbol{\omega}} - K\left(\Phi_{[1:n_o]}\Sigma_{\boldsymbol{\omega}}\right) \\ K &= \Sigma_{\boldsymbol{\omega}}\Phi_{[1:n_o]}^{\top}\left(\Sigma_{\xi}^{o} + \Phi_{[1:n_o]}\Sigma_{\boldsymbol{\omega}}\Phi_{[1:n_o]}^{\top}\right)^{-1}. \end{cases} \tag{A9}$$