Robot learning by observing humans activities and modeling failures.

Stefano Michieletto[1] and Alberto Rizzi[1] and Emanuele Menegatti[1]

Abstract— **People needs are varied and different. Service robotics aims to help people to satisfy these needs, but not all the feasible programs can be preloaded into a robot. The robot have to learn new tasks depending on the circumstances. A solution to this challenge could be Robot Learning from Demonstration (RLfD) [3] [2].**

In this paper, a RLfD framework is described in its entire pipeline. The data are acquired from a low cost RGB-D sensor, so the user can act naturally with no need of additional hardware. The information are subsequently elaborated to adapt to the robot structure and modeled to overcome the differences between human and robot.

Experiments are performed using input data coming from a publicly available dataset of human actions, and a humanoid robot, an Aldebaran NAO, is shown to successfully replicate an action based on human demonstrations and some further trials automatically generated from the learned model.

I. Introduction

Service robotics is asked to help people in everyday life: from carrying heavy stuff to opening a door, from taking care of elderly to playing games with kids. The requested tasks can vary a lot from one person to another and preparing the robot to any possible human need is not a feasible strategy. New skills should be learned directly from human being.

Robot Learning From Demonstration (RLfD) [3] [2] also called Imitation Learning is a programming paradigm that uses demonstrations in order to make a robot learn new tasks. Several approaches were adopted in RLfD: Schaal et al. [20] used motion primitives to encode learning data, Akgun et al. [1] extracted keyframes to correctly model a skill, Calinon et al. [6] proposed an Hidden Markov Model/Gaussian Mixture Regression technique to reproduce human demonstrations in a multiple constraints environment.

As highlighted in [15], state of the art approaches are based on expensive or sophisticated hardware normally inappropriate for household applications. In our work, we used a low-cost RGB-D sensor to provide data to an improved version of the Donut Mixture Model (DMM) proposed by Grollman in [11].

The model adopted is suitable in case of failed demonstrations due to the mapping between human and robot joints. In fact, this operation can generate a failed robot attempt, even starting from a successful human example.

The estimated model can also be updated using the attempts performed by the robot: this feature is very important to rapidly obtain correct robot trajectories using only few human demonstrations.

[1]Stefano Michieletto and Alberto Rizzi and Emanuele Menegatti are with Department of Information Engineering (DEI), University of Padova, Via Gradenigo 6/B, 35131 Padova, Italy {michieletto, rizzialb, emg}@dei.unipd.it

Figure 1 shows an overview of the complete framework.

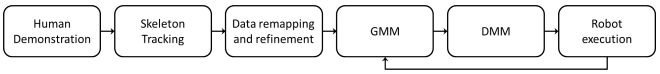


Fig. 1. Overview of the entire system proposed in this paper.

The robot used in this work is a very diffuse humanoid created for both scientific and service purposes: the Aldebaran NAO.

To test our RLfD framework a simple task involving only a 1 degree of freedom (DoF) has been learned: throwing a ball into a basket. The selected joint belongs to the upper part of the robot in order to avoid stability problems and focus our work to the learning system. The demonstrations used to train the framework come from a public action dataset, and the actors even do not know they are teaching to a robot.

The remainder of the paper is organized as follows: in Section II, the data acquisition system is presented. The robot structure and the joints considered in this work are illustrated in Section III. An analysis of the considered human motions and the data refinement process are described in Section IV. The Robot Learning from Demonstration framework is analyzed in Section V. The tests performed and the results collected are reported in Section VI. Conclusions and future works are contained in Section VII.

II. Data acquisition

Different data acquisition systems can be used to convey the demonstrations from users to the robot: motion sensors [5], kinesthetic teaching [13], or vision systems [8].

Motion sensors techniques imply to put some stuff on the user, while kinesthetic demonstrations involve robot motion guiding by a human performer. A naive user hardly suits to these methods, because they are unnatural or uncomfortable. We propose to use a RGB-D sensor [12] able to acquire the scene at 30 fps with a resolution of 640x480 pixels for both color and depth information. These sensors are cheap, low weight and power consuming, so they can be easily mounted on a mobile robot for service robotics purposes.

Recently, a large variety of computer vision algorithms using RGB-D sensors has been developed to estimate human pose [23], perform skeleton tracking [14] and recognize actions and activities [16]. We used a skeleton tracking algorithm [19] running at 30 fps to acquire joint positions and orientations (Figure 2) subsequently elaborated to understand the motion and guide the robot in the learning phase.

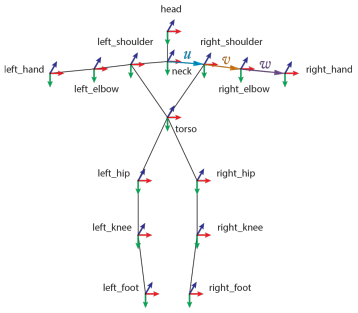


Fig. 2. Skeleton joints provided by the tracker.

III. Robot Structure Analysis

The robot used in this work is an Aldebaran NAO H25 v 4.0. It is a small humanoid robot with 25 DoF and an integrated Intel Atom CPU @ 1.6GHz. NAO comes with two gyroscopes, an accelerometer, a feedback provided from all its joints and pressure sensors on its feet. It can communicate with external systems using a Wi-Fi connection. A complete software suite and a SDK[1] package are also provided to fully program the humanoid platform and interact with the NAOqi application programming interface (API).

NAO is one of the most popular humanoid robots in the market, it is used for both research and service purposes and for these reasons we chose it to test our algorithms, but the following considerations can be easily adapted to any other suitable robot.

The remainder of this work involves only the upper-body motion in order to avoid stability problems, as we said before.

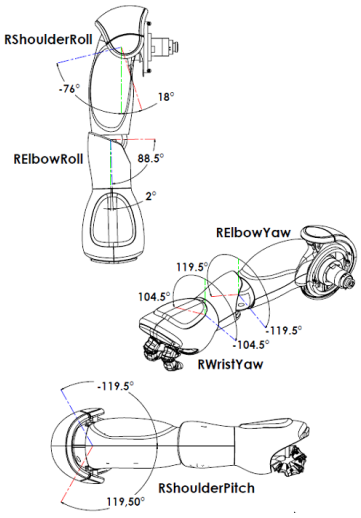


Fig. 3. Aldebaran NAO, right arm working area.

Starting from the skeleton data, we extract the information relative to the right arm movement. This information can be used to properly control the robot arm (Figure 3)[2]. Our goal is to extract the most significant joint from the movement performed by humans and remap it to the robot. Unluckily, we do not know the most informative joint at priori, so we should analyze the data from all the joints we are able to acquire.

Shoulder Roll: The right shoulder roll angle α is computed using the normalized scalar product of vector $u\,(neck, r\_soulder)$, starting from the neck skeleton joint and ending to the right shoulder skeleton joint, and the vector $v\,(r\_soulder, r\_elbow)$, starting from the right shoulder skeleton joint and ending to the right elbow skeleton joint. In Equation 1 the complete analytic formula is reported.

$$\alpha = \frac{\pi}{2} - \arccos\left(\frac{u \cdot v}{\|u\|\|v\|}\right) \tag{1}$$

Shoulder Pitch: The right shoulder pitch angle β is calculated projecting the vector $v\,(r\_soulder, r\_elbow)$, starting from the right shoulder skeleton joint and ending to the right elbow skeleton joint, onto the XY plane, and then using the formula in Equation 2.

$$\beta = \begin{cases} \arccos\left(\frac{y_v}{\mathcal{P}_{XY}(v)}\right) & if \quad z_v < 0; \\ \pi - \arccos\left(\frac{y_v}{\mathcal{P}_{XY}(v)}\right) & if \quad z_v \geq 0, y_v \geq 0; \\ -\pi - \arccos\left(\frac{y_v}{\mathcal{P}_{XY}(v)}\right) & if \quad z_v \geq 0, y_v < 0. \end{cases} \tag{2}$$

where y_v and z_v are respectively the y and z components of the considered vector v.

Elbow Roll: In order to compute the right elbow roll angle γ, the vector $-v\,(r\_soulder, r\_elbow)$, from the right elbow skeleton joint to the right shoulder skeleton joint, and the vector $w\,(r\_elbow, r\_hand)$, from the right elbow skeleton joint to the right hand skeleton joint, are treated in a similar way than in Equation 1. The resulting formula is reported in Equation 3

$$\gamma = \pi - \arccos\left(\frac{-v \cdot w}{\|v\|\|w\|}\right) \tag{3}$$

Elbow Yaw: The right elbow yaw angle δ is calculated considering the y component of the vector $w\,(r\_elbow, r\_hand)$, starting from the right elbow skeleton joint and ending to the right hand skeleton joint. In Equation 4 the complete analytic formula is reported.

$$\delta = -\arcsin\left(y_w\right) \tag{4}$$

No data are available from the skeleton tracker to properly calculate the wrist rotations.

IV. Data Refinement

The analysis of the considered motions highlights that the human elbow is the most significant joint we can track properly. On the other hand the robot forearm is very short and the throws are consequently ineffective (the maximum distance reachable is 5 cm). The solution adopted remaps the human elbow roll angle to the robot shoulder pitch angle in order to keep the task simple (only a joint is involved) and make the robot able to throw the ball over 60 cm.

[1] *http://www.aldebaran-robotics.com/documentation/dev/sdk.html*

[2] More information can be found at *http://www.aldebaran-robotics.com/documentation/family/nao_h21/joints_h21.html*

Human joints are more "advanced" than the robot ones: the angle values can overcome the limits of the robot and the movements vary a lot from one person to another. Some thresholds were introduced to clearly identify the action to learn. All the data before a starting angle and after an ending angle are automatically discarded. Finally the angle distance between the maximum and minimum angle of each demonstration are properly rescaled from the algorithm to fulfill the robot joint limits.

V. The Learning System

The human motions remapped to the robot joints and subsequently refined can now be executed by the NAO, but an initially correct attempt could become a failed trial to the robot. The proposed solution implements a model suitable for failure demonstration called Donut Mixture Model.

A. Donut Mixture Model

Donut Mixture Model (DMM) is a probability manner to encode a set of data, in this work demonstrations. The input dataset $X = \{x_n\}_{n=1}^{N}$ is a set of N trajectories $x_n = \{\xi_{n,t}\dot{\xi}_{n,t}\}_{t=1}^{T_n}$, of possibly different dimensions T_n. Each trajectory is composed by joint positions $\xi = \{\xi_j\}_{j=1}^{D}$ and velocities $\dot{\xi} = f_\theta(\xi) = \{\dot{\xi}_j\}_{j=1}^{D}$, where D is the number of joints involved in the trajectory. The DMM is used to approximate the non-linear function $\dot{\xi} = f_\theta(\xi)$.

A Donut component in the mixture model is defined as the difference between two Gaussian components:

$$\mathcal{D}(x|\mu_\alpha, \mu_\beta, \Sigma_\alpha, \Sigma_\beta, \gamma) = \gamma\mathcal{N}(x|\mu_\alpha, \Sigma_\alpha) - (\gamma-1)\mathcal{N}(x|\mu_\beta, \Sigma_\beta) \tag{5}$$

where

- $\mathcal{D}$ is the Donut distribution;
- x is a trajectory;
- $\mathcal{N}$ is a standard normal distribution;
- μ_α and Σ_α are respectively the mean vector and the covariance matrix of the first Gaussian;
- μ_β and Σ_β are respectively the mean vector and the covariance matrix of the second Gaussian;
- $\gamma > 1$.

The aim of a Donut distribution is looking for a good trajectory in the covariance space between the two Gaussian distributions. In order to explore this space the distribution has to avoid the best approximation of the collected demonstrations obtained from a Gaussian distribution. Therefore the mean of the two Gaussians should be the same, $\mu_\alpha = \mu_\beta$. In order to keep constant the shape of the covariance, a parameterization is applied to the Donut covariance matrices, such that $\Sigma_\alpha = \Sigma/r_\alpha^2$ and $\Sigma_\beta = \Sigma/r_\beta^2$.

The Donut distribution changes as follow:

$$\mathcal{D}(x|\mu, \Sigma, r_\alpha, r_\beta, \gamma) = \gamma\mathcal{N}(x|\mu, \Sigma/r_\alpha^2) - (\gamma-1)\mathcal{N}(x|\mu, \Sigma/r_\beta^2) \tag{6}$$

In order to calculate the Gaussian distributions, a Gaussian Mixture Model (GMM) is computed from the input data. A weighted Expectation Maximization (EM) [18] process is used to iteratively adjust the (π_k, μ_k, Σ_k) parameters of the K Gaussians composing the mixture, namely the priors vector, the mean vector and the covariance matrix. The Bayesian Information Criterion (BIC) [21] was used to estimate the optimal number of Gaussian components K in the Mixture. The score assigned at each considered K is described in Equation 7

$$S_{BIC} = -\mathcal{L} + \frac{n_p}{2}\log N \tag{7}$$

where

- $\mathcal{L} = \sum_{j=1}^{N}\log\left(p\left(\xi_j\right)\right)$ is the model log-likelihood;
- $n_p = (K-1) + K(D + 1/2 D(D+1))$ is the number of free parameters required for a mixture of K components with full covariance matrix.

The log-likelihood measures how well the model fits the data, while the second term aims to avoid data overfitting to keep the model general. The overall state of the GMM is described by $\theta = \{K, \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}\}$.

The probability that a joint assumes a certain velocity given its position is

$$p(\dot{\xi}|\xi) = \sum_{k=1}^{K}\tilde{\pi}_k\,\mathcal{D}(\dot{\xi}|\tilde{\mu}_k, \tilde{\Sigma}_k, \varepsilon) \tag{8}$$

$$\tilde{\mu}_k = \tilde{\mu}_k(\xi) = \mu_k(\dot{\xi}) + \Sigma_k(\dot{\xi}, \xi)\Sigma_k^{-1}(\xi, \xi)(\xi - \mu_k(\xi))$$

$$\tilde{\Sigma}_k = \Sigma_k(\dot{\xi}, \dot{\xi}) + \Sigma_k(\dot{\xi}, \xi)\Sigma_k^{-1}(\xi, \xi)\Sigma_k(\xi, \dot{\xi})$$

$$\tilde{\pi}_k = \tilde{\pi}_k(\xi) = \frac{\pi_k\mathcal{N}(\xi; \mu_k(\xi), \Sigma_k(\xi, \xi))}{\sum_{k=1}^{K}\pi_k\mathcal{N}(\xi; \mu_k(\xi), \Sigma_k(\xi, \xi))}$$

$$\varepsilon = 1 - \frac{1}{1 + \|\tilde{V}[\dot{\xi}, \xi, \theta]\|}$$

$$\tilde{V}[\dot{\xi}|\xi, \theta] = -\tilde{E}[\dot{\xi}|\xi, \theta]\tilde{E}[\dot{\xi}|\xi, \theta]^\top + \sum_{k=1}^{K}\pi_k(\mu_k\mu_k^\top + \Sigma_k)$$

where

- γ is set to 2, so in Equation 6 the first Gaussian is doubled and than the second Gaussian is subtracted;
- $\tilde{\mu}_k$ is the k-th mean vector derived from the estimated GMM;
- $\tilde{\Sigma}_k$ is the k-th covariance matrix derived from the estimated GMM;
- $\tilde{\pi}_k$ is the k-th mean vector derived from the estimated GMM, with $\sum_{k=1}^{K}\pi_k = 1$
- ε is the (exploration) exploratory factor, with $0 \leq \varepsilon \leq 1$, so we have maximum exploration (high variability) for $\varepsilon \to 1$, while the minimum exploration (low variability) is reached for $\varepsilon \to 0$;
- $\tilde{V}[\dot{\xi}|\xi, \theta]$ is the overall variance of the estimated GMM connecting the variability generated by human demonstrators to the system exploration.

A novel trajectory can be generated from the computed DMM. If this new trajectory fails in reaching the task, the model should be updated reiteratively in order to get success. A first technique consists of recomputing the model using the EM process on the entire dataset plus the generated trajectory. This method grows in time and memory proportionally to the input dataset. An alternative uses a sample and merge approach to maintain constant both time and memory.

B. Optimization

The new trajectory generation requires that the DMM probability density function is maximized. Differently from GMM, DMM has no analytical solution to the problem, so an optimization technique should be used. Grollman [11] proposed gradient ascendant (steepest gradient) to find a maximum around an initial guess. The result could be both global or local maxima depending on the starting point.

In this paper the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [4] [9] [10] [22] is used to overcame some problems we noticed using gradient ascendant that is very inefficient when the function to be maximized has long narrow valleys. BFGS is a quasi-Newton optimization state-of-the-art method that iteratively approximates Hessian matrix to converge to the solution.

Given f convex, continuously differentiable function to maximize, x_0 starting point in the f domain and B_0 initial approximation of the Hessian matrix set to any symmetric positive definite matrix (i.e. identity matrix), at each iteration k the point x_k and the matrix B_k is updated using the formulas:

$$x_{k+1} = x_k + t\Delta x$$

$$B_{k+1} = B_k - \frac{B_k s_k s_k^\top B_k}{s_k^\top B_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k} \tag{9}$$

where

- t is the step size;
- $\Delta x = -B_{k-1}^{-1} \nabla f(x_{k-1})$ is the direction to move to find the maxima, it is also used as stopping criterion in the case it is lower than a predefined threshold;
- $s_k = x_k - x_{k-1}$;
- $y_k = \nabla f(x_k) - \nabla f(x_{k-1})$.

This method assures fast convergence, it avoids calculation of second-order information, that is computationally expensive using numerical differentiation. BFGS is a general purpose method, very effective in several situations, and with a minimal variation also with large scale applications. Table I shows a comparison between a gradient ascendant implementation and the BFGS algorithm we proposed. Three initial points were considered on a DMM generated randomly. The convergence of the algorithm, the number of iterations, the resulting point and its corresponding value in the Donut are considered.

The result still could be a local maximum, this is an intrinsic problem due to use of gradient as moving direction. It mainly depends from the designed starting point, at example if it is near to a local maximum. Another typical error could happen when the curvature near to the starting point is lower than the threshold and the algorithm stops at the first iteration even if the selected point is in the Donut tail.

In order to avoid these situations, we introduced two polices:

- *Start Check (SC)*: the starting point is changed if the neighborhood presents a gradient already lower than the selected threshold;
- *Multi-way Search (MW)*: the research of the maximum is reiterated starting from S different initial points simultaneously. The number of starting points S can vary with the number of processors and the response time requested by the application. In this work we set $S = K$, and the starting points are the means of the K Donut components. Finally, we considered the overall maximum, thus only the best resulting maximum is used. This policy allows us to avoid local maxima that lead to sub-optimal trajectories.

In Table II three different tests are reported in which *SC* policy is applied to non converging or local converging BFGS instances. Table III shows the *MW* policy with $S = 4$ compared with two different standard BFGS iterations. For both Table II and Table III the same DMM generated for the Table I comparison was used as function to maximize.

VI. Experimental Results

Our approach is tested using the skeleton data provided in the *Throw Over Head* action from the IAS-Lab Action Dataset[3] [17] [16], in which 12 actors threw a ball using a basketball approach (Figure 4). The arm is raised, the elbow is bent, the limb is extended, and the wrist is rotated so that the ball reaches the target; the entire movement is correctly repeated 3 times for each actor. The acquired data were refined as described in Section IV and remapped to the robot joint space. From the original 36 trajectories, we removed 2 samples in which the number of skeleton joint data provided is too low to be effective in the robot movement.

These demonstrations are the input to our Learning from Demonstration framework. The aim is to make an Aldebaran NAO learn how to play basketball. The robot should be able to put the ball into a basket placed 40 cm in front of it (Figure 5). The results of the system are compared with the output trajectories computed using well-know GMM/GMR framework [7] suitable for robot learning when the task is successfully performed by the demonstrators. As in the original article, the means of the Gaussian components are used to generate the trajectory from the system.

Fig. 5. Overview of experimental scenario: the NAO is placed at 40 cm from the basket.

Looking at our input trajectories we notice, as we expected, that only few (11 over 34) throws enter correctly into the basket, the remaining 23 throws score no points; even if

[3]The IAS-Lab Action Dataset is publicly available at *http://robotics.dei.unipd.it/actions*

TABLE I

COMPARISON BETWEEN BFGS AND GRADIENT ASCENDANT OPTIMIZATION ALGORITHMS.

Method	BGFS	Grad. Asc.	BGFS	Grad. Asc.	BGFS	Grad. Asc.
Convergence	Yes	No	Yes	Yes	Yes	No
Iterations	4	250	2	194	5	250
x_0	-0.02	-0.02	0	0	0.02	0.02
x_{max}	-0.0029	-0.0029	-0.0029	-0.0029	-0.0029	-0.0029
$f(x_{max})$	114.23	114.23	114.23	114.23	114.23	114.23

TABLE II

COMPARISON BETWEEN STANDARD BFGS AND BFGS PLUS SC POLICY ON NON CONVERGING OR LOCAL CONVERGING STARTING POINTS.

Method	BFGS	BFGS + SC	BFGS	BFGS + SC	BFGS	BFGS + SC
Convergence	No	Yes	Yes	Yes	Yes	Yes
Local/Global	-	G	L	G	L	G
Iterations (SC)	1	4 (8)	2	5 (3)	5	3 (5)
x_0	-0.25	-0.031	-0.063	-0.034	-0.114	0.006
$\nabla f(x_0)$	0	1.25	-2.48 E -19	0.027	0	-197.3
x_{max}	-0.25	-0.0029	-0.053	-0.0029	-0.10	-0.0029
$f(x_{max})$	0	114.23	-1.47 E -17	114.23	-1.73 E -84	114.23

TABLE III

COMPARISON BETWEEN STANDARD BFGS AND BFGS PLUS MW POLICY.

Method	BFGS	BFGS	MW 1	MW 2	MW 3	MW 4
Convergence	No	Yes	Yes	Yes	Yes	Yes
Local/Global	-	G	L	G	G	G
Iterations	1	5	2	3	2	5
x_0	-0.07	-0.038	-0.063	-0.0062	-0.001	0.024
$\nabla f(x_0)$	0	7.29 E -6	-2.48 E -19	2597.8	-2026.36	0.24
x_{max}	-0.07	-0.0029	-0.053	-0.0029	-0.0029	-0.0029
$f(x_{max})$	0	114.23	-1.47 E -17	114.23	114.23	114.23

Fig. 4. Screen-shots from the *Throw Over Head* action performed by different actors in the IAS-Lab Action Dataset.

all the 34 demonstrations were successfully performed by the actors. This result is due to difference between the human and the robot DoFs, and, as we said before, we chose a Learning from Failure Demonstrations framework to model human observations mainly for this reason. It is also worth to notice that the difference is further augmented by the noise introduced by the sensor.

We divided the 34 demonstrations into three groups:

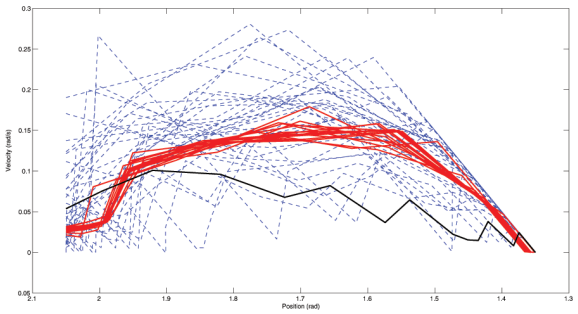
- successful and failed demonstrations (34 trajectories)
- failed demonstrations only (23 trajectories)
- successful demonstrations only (11 trajectories)

for each of these groups we generated 15 new trajectories coming from 15 different DMM, where the 1^{st} DMM models the initial dataset (34, 23 or 11 demonstrations), the 2^{nd} DMM models the initial dataset plus the previously generated trajectory, the n^{th} DMM models the initial dataset plus the $(n-1)^{th}$ trajectories generated previously. Differently than in Grollman's work, where the generation is stopped when a successful trajectory comes out, we preferred to generate all the 15 trajectories in order to observe the system behavior even with a high number of correct demonstrations.

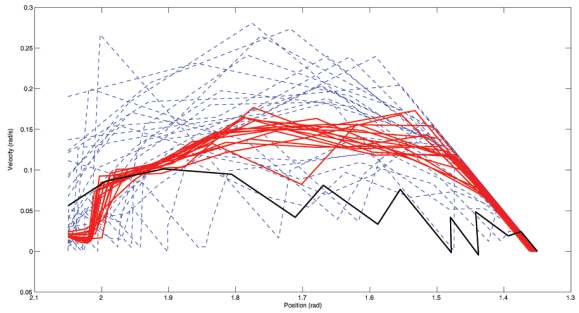
In the first group, the input trajectories are both correct and incorrect. The generated trajectories are showed in Figure 6 (a), the 15 trajectories from our framework are drawn in red; in black the one generated from the GMM/GMR framework; the input trajectories are dotted in blue. The 5th, 6th, 13th and 15th trajectories scored a point, while the one from GMM/GMR did not.

The results from the second group are showed in Figure 6 (b). As before the 23 incorrect input trajectories are dotted in blue; in red the ones from our framework; the trajectory generated by the GMM/GMR framework is black. Since this group is composed by failed demonstrations only, we expected an improvement in our framework performances and still a bad throw from GMM/GMR framework. As we expected, only 5 generated trajectories failed, the 2nd, the 6th, the 8th, the 13th and the one from GMM/GMR.

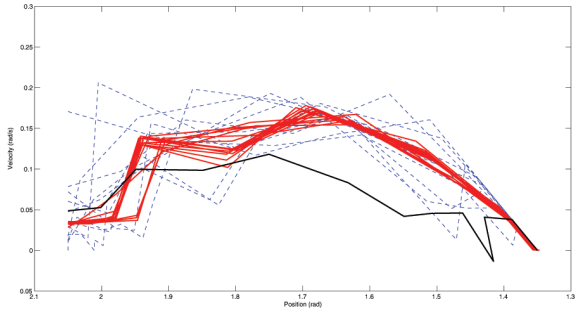
Figure 6 (c) shows the third group results. As usual the 11



(a) Generated trajectories from both successful and failed demonstrations.



(b) Generated trajectories from failed demonstrations only.



(c) Generated trajectories from successful demonstrations only.

Fig. 6. Results obtained from the demonstrations collected from human observations (blue dots) using our framework (in red) and a GMM/GMR framework (black).

successful trajectories are dotted in blue; the ones in red are generated from our framework; the GMM/GMR trajectory is in black. Differently from what we thought, our framework overcomes the GMM/GMR one even in this group. In fact, the GMM/GMR throw touches the basket border, but finally goes out. On the other hand, our framework generated only 2 failed trajectories: the 1st and the 11th.

VII. Conclusions

In this paper, a RLfD framework was developed in order learn a new skill from human observations. The model used is suitable to overcome the differences between the human and robot motions, with only few more trials performed by the robot. We compared the results with a different RLfD technique. Our framework generates better trajectories in all the considered scenarios.

As future work we will focus on time performances in order to make it work real-time. We also plan to increase the number of robot joints controlled by the framework, and for this reason we improved the model proposed by Grollman [11]. In fact, some preliminary tests showed that our approach does not incur the problems highlighted by Grollman in his work.

References

[1] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 2012.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 2009.

[3] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics*, 2008.

[4] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 1970.

[5] S. Calinon and A. Billard. Stochastic gesture production and recognition model for a humanoid robot. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 2004.

[6] S. Calinon, F. D'halluin, D. G. Caldwell, and A. G. Billard. Handling of multiple constraints and motion alternatives in a robot programming by demonstration framework. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, 2009.

[7] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 2007.

[8] R. Dillmann. Teaching and learning of robot tasks via observation of human performance. *Robotics and Autonomous Systems*, 2004.

[9] R. Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[10] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 1970.

[11] D. H. Grollman and A. G. Billard. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 2012.

[12] J. Han, L. Shao, D. Xu, and J. Shotton. Enhanced computer vision with microsoft kinect sensor: A review. *IEEE Transactions on Cybernetics*, 2013.

[13] M. Hersch, F. Guenter, S. Calinon, and A. Billard. Dynamical system modulation for robot learning via kinesthetic demonstrations. *Robotics, IEEE Transactions on*, 2008.

[14] A. Kar. Skeletal tracking using microsoft kinect. *Methodology*, 2010.

[15] A. León, E. F. Morales, L. Altamirano, and J. R. Ruiz. Teaching a robot to perform task through imitation and on-line feedback. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, 2011.

[16] M. Munaro, G. Ballin, S. Michieletto, and E. Menegatti. 3D flow estimation for human action recognition from colored point clouds. *Journal on Biologically Inspired Cognitive Architectures*, 2013.

[17] M. Munaro, S. Michieletto, and E. Menegatti. An evaluation of 3D motion flow and 3D pose estimation for human action recognition. In *Learning in Graphical Models*, 2013.

[18] R. Neal, and G.E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *RSS Workshops: RGB-D: Advanced Reasoning with Depth Cameras*, 1998.

[19] Nite middleware [online] *http://www.primesense.com/solutions/nite-middleware*.

[20] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In *Robotics Research*, 2005.

[21] G. Schwarz. Estimating the dimension of a model. *The annals of statistics*, 1978.

[22] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 1970.

[23] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 2013.