2010 IEEE-RAS International Conference on Humanoid Robots
Nashville, TN, USA, December 6-8, 2010

Learning the skill of archery by a humanoid robot iCub

Petar Kormushev[1], Sylvain Calinon[2], Ryo Saegusa[3], Giorgio Metta[4]

***Abstract*— We present an integrated approach allowing the humanoid robot iCub to learn the skill of archery. After being instructed how to hold the bow and release the arrow, the robot learns by itself to shoot the arrow in such a way that it hits the center of the target. Two learning algorithms are proposed and compared to learn the bi-manual skill: one with Expectation-Maximization based Reinforcement Learning, and one with chained vector regression called the ARCHER algorithm. Both algorithms are used to modulate and coordinate the motion of the two hands, while an inverse kinematics controller is used for the motion of the arms. The image processing part recognizes where the arrow hits the target and is based on Gaussian Mixture Models for color-based detection of the target and the arrow's tip. The approach is evaluated on a 53-DOF humanoid robot iCub.**

I. INTRODUCTION

Acquiring new motor skills involves various forms of learning. The efficiency of the process lies in the interconnections between imitation and self-improvement strategies. Similarly to humans, a robot should ideally be able to acquire new skills by employing such mechanisms.

Some tasks can be successfully transferred to the robot using only imitation strategies [1] [2]. Other tasks can be learned very efficiently by the robot alone using Reinforcement Learning (RL) [3]. The recent development of compliant robots progressively moves their operational domain from industrial applications to home and office uses, where the role and tasks can not be determined in advance. While some tasks allow the user to interact with the robot to teach it new skills, it is generally preferable to provide a mechanism that permits the robot to learn to improve and extend its skills to new contexts under its own guidance.

Researchers in machine learning and robotics have made tremendous efforts and advances to move RL algorithms from discrete to continuous domains, thus extending the possibilities for robotic applications. Until recently, policy gradient algorithms (such as Episodic REINFORCE [4] and Episodic Natural Actor-Critic eNAC [5]) have been a well-established approach to cope with the high dimensionality [6]. Unfortunately, they also have shortcomings, such as high sensitivity to the learning rate.

To avoid such problems, Kober *et al* proposed in [7] an episodic RL algorithm called Policy learning by Weighting Exploration with the Returns (PoWER). It is based on Expectation-Maximization algorithm (EM) and one major advantage over policy-gradient-based approaches is that it does not require a learning rate parameter. This is desirable because tuning a learning rate is usually difficult to do for control problems but critical for achieving good performance of policy-gradient algorithms. PoWER also demonstrates superior performance in tasks learned directly on a real robot, such as the ball-in-a-cup task [8] and the pancake flipping task [9].

Authors {1,2} are with the Advanced Robotics Department and authors {3,4} are with the Robotics, Brain and Cognitive Sciences Department of the Italian Institute of Technology (IIT), 16163 Genova, Italy. {petar.kormushev,sylvain.calinon,ryo.saegusa,giorgio.metta}@iit.it.

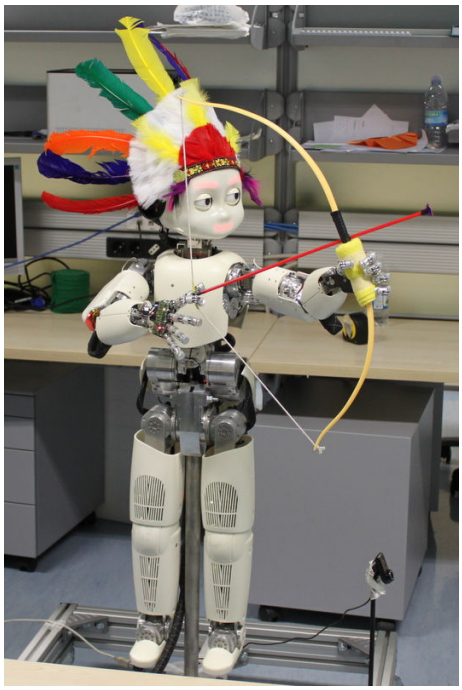


Fig. 1. Experimental setup for the archery task. A position-controlled 53-DOF humanoid robot iCub learns to shoot arrows using a bow and learn to hit the center of the target using RL algorithm and visual feedback from a camera.

In this paper we present an integrated approach allowing a humanoid robot to learn the skill of archery. After being instructed how to hold the bow and release the arrow, the robot learns by itself to shoot the arrow in such a way that it hits the center of the target. The archery task was selected because: (1) it involves bi-manual coordination; (2) it can be performed with slow movements of the arms and using small torques and forces; (3) it requires using tools (bow and arrow) to affect an external object (target); (4) it is an appropriate task for testing different learning algorithms and aspects of learning, because the reward is inherently defined by the high-level description of the task goal; (5) it involves integration of image processing, motor control and learning parts in one coherent task.

The focus of the paper is on learning the bi-manual

(a) (b)

Fig. 2. Archery by a human and automata. (a) the shooting form in Kyudo [10]. (b) the replica of the archery automata [11].

coordination necessary to control the shooting direction and velocity in order to hit the target. Two learning algorithms are proposed and compared: one with Expectation-Maximization based Reinforcement Learning, and one with chained vector regression. Both algorithms are used to modulate and coordinate the motion of the two hands, while inverse kinematics controller is used for the motion of the arms. We propose solutions to the learning part, the image processing part used to detect the arrow's tip on the target, and the motor control part of the archery training.

II. DESCRIPTION OF THE ARCHERY TASK

Archery is the art of propelling arrows with the use of a bow and has been developed to high levels in many societies. In North America, archery was well known among native people from pre-Columbian times. In Japan archery is known as Kyudo, shown in Fig. 2 (a), which is recognized as a rather mental sport, with a longer bow and simpler equipment than in European archery [10].

At the end of the 19th century, H. Tanaka created an archery automata which was able to perform the complete sequential movements to shoot an arrow [11]. The movements were composed of four primitives: grasping an arrow, setting the arrow at the bow string, drawing the bow, and releasing the shot. Fig. 2 (b) shows a replica of the archery automata. Instead of pulling the string of the bow with the right hand towards the torso, this automata is actually pushing the bow with the left hand in the opposite direction.

The independently developed examples of archery show that the same skill can be achieved in a different manner, and that the skill is adapted differently depending on the tool and embodiment, to achieve the same result. Similarly, in our robotic archery experiment, we needed to adapt the setup and shooting movement to the specifics of our humanoid robot.

III. PROPOSED APPROACH

In this section we propose two different learning algorithms for the archery training and one image processing algorithm for detecting the arrow on the target. The focus of the proposed approach falls on learning the bi-manual coordination for shooting the arrow with a desired direction and velocity. Similarly to [12], we consider discrete goal-directed movements where the relative position between the two hands represents coordination patterns that the robot needs to learn. We do not consider the problem of learning how to grasp the bow or pull the arrow, and therefore these sub-tasks are pre-programmed.

A. Learning algorithm 1: PoWER

As a first approach for learning the bi-manual coordination needed in archery, we use the state-of-the-art EM-based RL algorithm PoWER by Jens Kober and Jan Peters [7]. We selected PoWER algorithm because it does not need a learning rate (unlike policy-gradient methods) and also because it can be combined with importance sampling to make better use of the previous experience of the agent in the estimation of new exploratory parameters.

PoWER uses a parameterized policy and tries to find values for the parameters which maximize the expected return of rollouts (also called trials) under the corresponding policy. For the archery task the policy parameters are represented by the elements of a 3D vector corresponding to the relative position of the two hands performing the task.

We define the return of an arrow shooting rollout τ to be:

$$R(\tau) = e^{-||\hat{r}_T - \hat{r}_A||}, \tag{1}$$

where $\hat{r}_T$ is the estimated 2D position of the center of the target on the target's plane, $\hat{r}_A$ is the estimated 2D position of the arrow's tip, and $||\cdot||$ is Euclidean distance.

As an instance of EM algorithm, PoWER estimates the policy parameters θ to maximize the lower bound on the expected return from following the policy. The policy parameters θ_n at the current iteration n are updated to produce the new parameters θ_{n+1} using the following rule (as described in [8]):

$$\theta_{n+1} = \theta_n + \frac{\left\langle (\theta_k - \theta_n) R(\tau_k) \right\rangle_{w(\tau_k)}}{\left\langle R(\tau_k) \right\rangle_{w(\tau_k)}}. \tag{2}$$

In Eq. (2), $(\theta_k - \theta_n) = \Delta\theta_{k,n}$ is a vector difference which gives the relative exploration between the policy parameters used on the k-th rollout and the current ones. Each relative exploration $\Delta\theta_{k,n}$ is weighted by the corresponding return $R(\tau_k)$ of rollout τ_k and the result is normalized using the sum of the same returns. Intuitively, this update rule can be thought of as a weighted sum of parameter vectors where higher weight is given to these vectors which result in better returns.

In order to minimize the number of rollouts which are needed to estimate new policy parameters, we use a form of importance sampling technique adapted for RL [3] [7] and denoted by $\langle\cdot\rangle_{w(\tau_k)}$ in Eq. (2). It allows the RL algorithm to re-use previous rollouts τ_k and their corresponding policy parameters θ_k during the estimation of the new policy

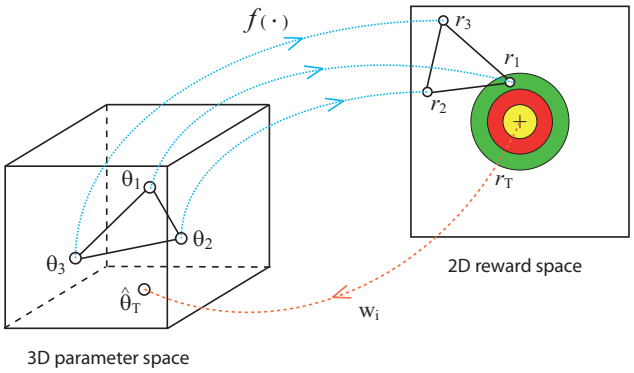


Fig. 3. The conceptual idea underlying the ARCHER algorithm.

parameters θ_{n+1}. The importance sampler is defined as:

$$\Big\langle f(\theta_k, \tau_k) \Big\rangle_{w(\tau_k)} = \sum_{k=1}^{\sigma} f(\theta_{\mathrm{ind}(k)}, \tau_{\mathrm{ind}(k)}), \tag{3}$$

where σ is a fixed parameter denoting how many rollouts the importance sampler is to use, and $\mathrm{ind}(k)$ is an index function which returns the index of the k-th best rollout in the list of all past rollouts sorted by their corresponding returns, i.e. for $k = 1$ we have:

$$\mathrm{ind}(1) = \underset{i}{\mathrm{argmax}}\, R(\tau_i), \tag{4}$$

and the following holds: $R(\tau_{\mathrm{ind}(1)}) \geq R(\tau_{\mathrm{ind}(2)}) \geq \ldots \geq R(\tau_{\mathrm{ind}(\sigma)})$. The importance sampler allows the RL algorithm to calculate new policy parameters using the top-σ best rollouts so far. This reduces the number of required rollouts to converge and makes this RL algorithm applicable to online learning.

B. Learning algorithm 2: ARCHER

For a second learning approach we propose a custom algorithm developed and optimized specifically for problems like the archery training, which have a smooth solution space and prior knowledge about the goal to be achieved. We will refer to it as the ARCHER algorithm (Augmented Reward CHainEd Regression). The motivation for ARCHER is to make use of richer feedback information about the result of a rollout. Such information is ignored by the PoWER RL algorithm because it uses scalar feedback which only depends on the distance to the target's center. ARCHER, on the other hand, is designed to use the prior knowledge we have on the optimum reward possible. In this case, we know that hitting the center corresponds to the maximum reward we can get. Using this prior information about the task, we can view the position of the arrow's tip as an augmented reward. In this case, it consists of a 2-dimensional vector giving the horizontal and vertical displacement of the arrow's tip with respect to the target's center. This information is obtained either directly from the simulated experiment in Section IV or calculated by the image processing algorithm in Section III-C for the real-world experiment. Then, ARCHER uses a chained local regression process that iteratively estimates new policy parameters which have a greater probability of leading to the achievement of the goal of the task, based on the experience so far.

Each rollout τ_i, where $i \in \{1, \ldots, N\}$, is initiated by input parameters $\theta_i \in \mathbb{R}^3$, which is the vector describing the relative position of the hands and is produced by the learning algorithms. Each rollout has an associated observed result (considered as a 2-dimensional reward) $r_i = f(\theta_i) \in \mathbb{R}^2$, which is the relative position of the arrow's tip with respect to the target's center $r_T = (0, 0)^T$. The unknown function f is considered to be non-linear due to air friction, wind flow, friction between the bow and the arrow, and etc. A schematic figure illustrating the idea of the ARCHER algorithm is shown in Fig. 3.

Without loss of generality, we assume that the rollouts are sorted in descending order by their scalar return calculated by Eq. 1, i.e. $R(\tau_i) \geq R(\tau_{i+1})$, i.e. that r_1 is the closest to r_T. For convenience, we define vectors $r_{i,j} = r_j - r_i$ and $\theta_{i,j} = \theta_j - \theta_i$. Then, we represent the vector $r_{1,T}$ as a linear combination of vectors using the N best results:

$$r_{1,T} = \sum_{i=1}^{N-1} w_i r_{1,i+1}. \tag{5}$$

Under the assumption that the original parameter space can be linearly approximated in a small neighborhood, the calculated weights w_i are transferred back to the original parameter space. Then, the unknown vector to the goal parameter value $\theta_{1,T}$ is approximated with $\hat{\theta}_{1,T}$ as a linear combination of the corresponding parameter vectors using the same weights:

$$\hat{\theta}_{1,T} = \sum_{i=1}^{N-1} w_i \theta_{1,i+1}. \tag{6}$$

In a matrix form, we have $r_{1,T} = WU$, where W contains the weights $\{w_i\}_{i=2}^{N}$, and U contains the collected vectors $\{r_{1,i}\}_{i=2}^{N}$ from the observed rewards of N rollouts. The least-norm approximation of the weights is given by $\hat{W} = r_{1,T}U^{\dagger}$, where $U^{\dagger}$ is the pseudoinverse of U.[1] By repeating this regression process when adding a new couple $\{\theta_i, r_i\}$ to the dataset at each iteration, the algorithm refines the solution by selecting at each iteration the N closest points to r_T. ARCHER can thus be viewed as a linear vector regression with a shrinking support region.

In order to find the optimal value for N (the number of samples to use for the regression), we have to consider both the observation errors and the function approximation error. The observation errors are defined by $\epsilon_\theta = ||\tilde{\theta} - \theta||$ and $\epsilon_r = ||\tilde{r} - r||$, where $\tilde{\theta}$ and $\tilde{r}$ are the real values, and θ and r are the observed values. The function approximation error caused by non-linearities is defined by $\epsilon_f = ||f - A\theta||$, where A is the linear approximation.

On the one hand, if the observations are very noisy ($\epsilon_r \gg \epsilon_f$ and $\epsilon_\theta \gg \epsilon_f$), it is better to use bigger values for N,

[1] In this case, we used a least-squares estimate. For more complex solution spaces, ridge regression or other regularization scheme can be considered.

in order to reduce the error when estimating the parameters w_i. On the other hand, for highly non-linear functions f ($\epsilon_f \gg \epsilon_r$ and $\epsilon_f \gg \epsilon_\theta$), it is better to use smaller values for N, i.e. to use a small subset of points which are closest to r_T in order to minimize the function approximation error ϵ_f. For the experiments presented in this paper we used $N = 3$ in both the simulation and the real-world, because the observation errors were kept very small in both cases.

The ARCHER algorithm can also be used for other tasks, provided that: (1) a-priori knowledge about the desired target reward is known; (2) the reward can be decomposed into separate dimensions; (3) the task has a smooth solution space.

C. Image processing algorithm

The problem of detecting where the target is, and what is the relative position of the arrow with respect to the center of the target, is solved by image processing. We use color-based detection of the target and the tip of the arrow based on Gaussian Mixture Model (GMM). The color detection is done in YUV color space, where Y is the luminance, and UV is the chrominance. Only U and V components are used to ensure robustness to changes in luminosity.

In a calibration phase, prior to conducting an archery experiment, the user explicitly defines on a camera image the position and size of the target and the position of the arrow's tip. Then, the user manually selects N_T pixels lying inside the target in the image, and N_A pixels from the arrow's tip in the image. The selected points produce two datasets: $c_T \in \mathbb{R}^{2\times N_T}$ and $c_A \in \mathbb{R}^{2\times N_A}$ respectively.

From the two datasets c_T and c_A, a *Gaussian Mixture Model* (GMM) is used to learn a compact model of the color characteristics in UV space of the relevant objects. Each GMM is described by the set of parameters $\{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$, representing respectively the *prior* probabilities, *centers* and *covariance* matrices of the model (full covariances are considered here). The *prior* probabilities π_k satisfy $\pi_k \in \mathbb{R}^{[0,1]}$ and $\sum_{k=1}^{K} \pi_k = 1$. A *Bayesian Information Criterion* (BIC) [13] is used to select the appropriate number of Gaussians K_T and K_A to represent effectively the features to track.

After each reproduction attempt, a camera snapshot is taken to re-estimate the position of the arrow and the target.[2] From the image $c_I \in \mathbb{R}^{2\times N_x \times N_y}$ of $N_x \times N_y$ pixels in UV color space, the center m of each object on the image is estimated through the weighted sum

$$m = \sum_{x=1}^{N_x}\sum_{y=1}^{N_y} \frac{1}{S_{x,y}} \sum_{k=1}^{K} \pi_k \, \mathcal{N}(c_{I,x,y};\ \mu_k, \Sigma_k) \begin{bmatrix} x \\ y \end{bmatrix}, \quad (7)$$

$$\text{with} \quad S_{x,y} = \sum_{j=1}^{K} \pi_j \, \mathcal{N}(c_{I,x,y};\ \mu_j, \Sigma_j).$$

[2]If the arrow did not stick to the wall, we put it back manually to the point of impact on the wall.

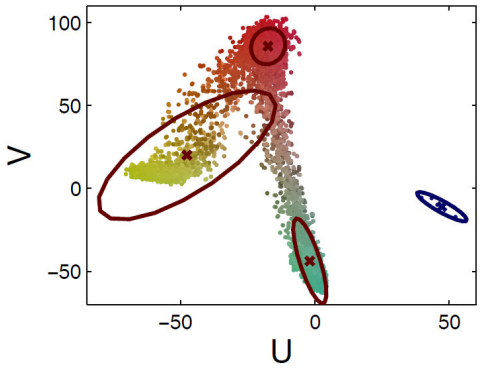


Fig. 4. Fitting a GMM to represent the target's and arrow's color characteristics in YUV color space. In this case, three Gaussians have been found to represent the target and a single Gaussian to represent the arrow.

In the above equation, $\mathcal{N}(c; \mu_k, \Sigma_k)$ is the probability defined by

$$\mathcal{N}(c; \mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^2 |\Sigma_k|}} \, e^{-\frac{1}{2}(c-\mu_k)^\top \Sigma_k^{-1}(c-\mu_k)}. \quad (8)$$

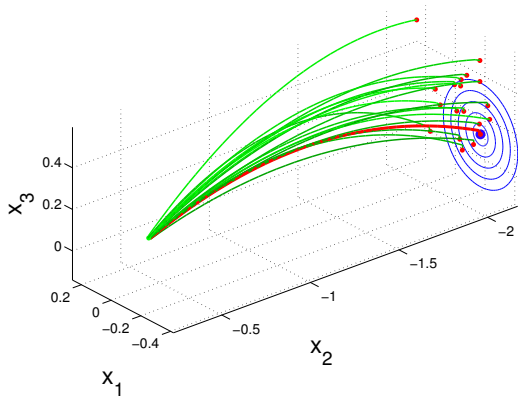
The reward vector is finally calculated as $r = m_T - m_A$, where m_A is the estimated center of the arrow and m_T is the estimated center of the target. Fig. 4 illustrates the work of the described algorithm with color data taken from an image of the real archery target.

IV. SIMULATION EXPERIMENT

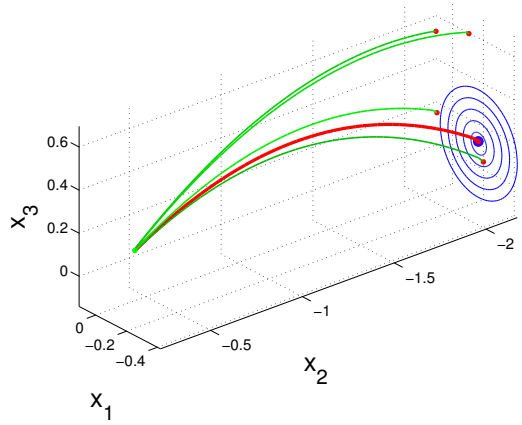
The two proposed learning algorithms (PoWER and ARCHER) are first evaluated in a simulation experiment. Even though the archery task is hard to model explicitly (e.g., due to the unknown parameters of the bow and arrow used), the trajectory of the arrow can be modeled as a simple ballistic trajectory, ignoring air friction, wind velocity and etc. A typical experimental result for each algorithm is shown in Fig. 5. In both simulations, the same initial parameters are used. The simulation is terminated when the arrow hits inside the innermost ring of the target, i.e. the distance to the center becomes less than 5 cm.

For a statistically significant observation, the same experiment was repeated 40 times with a fixed number of rollouts (60) in each session. The averaged experimental result is shown in Fig. 6. The ARCHER algorithm clearly outperforms the PoWER algorithm for the archery task. This is due to the use of 2D feedback information which allows ARCHER to make better estimations/predictions of good parameter values, and to the prior knowledge concerning the maximum reward that can be achieved. PoWER, on the other hand, achieves reasonable performance despite using only 1D feedback information.

Based on the results from the simulated experiment, the ARCHER algorithm was chosen to conduct the following real-world experiment.



(a) PoWER



(b) ARCHER

Fig. 5. Simulation of archery. Learning is performed under the same starting conditions with two different algorithms. The red trajectory is the final rollout. (a) PoWER algorithm needs 19 rollouts to reach the center. (b) ARCHER algorithm needs 5 rollouts to do the same.

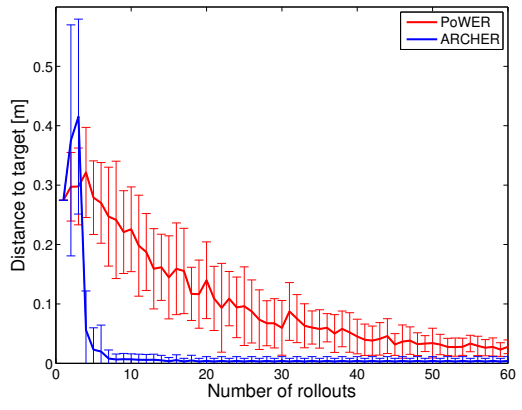


Fig. 6. Comparison of the speed of convergence for the PoWER and ARCHER algorithms. Statistics are collected from 40 learning sessions with 60 rollouts in each session. The first 3 rollouts of ARCHER are done with large random exploratory noise, which explains the big variance at the beginning.

Fig. 7. Real-world experiment using the iCub [14]. The distance between the target and the robot is 2.2 meters. The diameter of the target is 50 cm.

V. ROBOT EXPERIMENT

A. Robotic platform

The real-world robot experimental setup is shown in Fig.7. The experiment was conducted with iCub [14]. The iCub is an open-source robotic platform with dimensions comparable to a three and a half year-old child (about 104cm tall), with 53 degrees of freedom (DOF) distributed on the head, torso, arms, hands, and legs [15] [16]. Software modules in the architecture are interconnected using YARP [17] [18].

In the experiment, we used the torso, arms, and hands. The torso has 3 DOF (yaw, pitch, and roll). Each arm has 7 DOF, three in shoulder, one in the elbow and three in the wrist. Each hand of the iCub has 5 fingers and 19 joints although with only 9 drive motors several of these joints are coupled. We manually set the orientation of the neck, eyes and torso of the robot to turn it towards the target. The finger positions of both hands were also set manually to allow the robot to grip the bow and release the string suitably. We used one joint in the index finger to release the string. It was not possible to use two fingers simultaneously to release the string because of difficulties with synchronizing their motion. The posture of the left arm (bow side) was controlled by the proposed system, as well as the orientation of the right arm (string side). The position of the right hand was kept within a small area, because the limited range of motion of the elbow joint did not permit pulling the string close to the torso.

B. Robot control

To control the robot, the inverse kinematics module developed by Pattacini *et al* is used [19]. The proposed solution is to treat the inverse kinematics as an optimization under inequality constraints problem. Compared to standard Jacobian-based approaches, this approach has the advantage that the solver internally encapsulates knowledge of the joint bounds. Moreover, it is capable of dealing with a complex set of non-linear constrains expressed both in joint and task space. Another difference from standard Jacobian-based approaches is that this solution gives priority to the position in Cartesian space over the orientation. Hence, it is possible

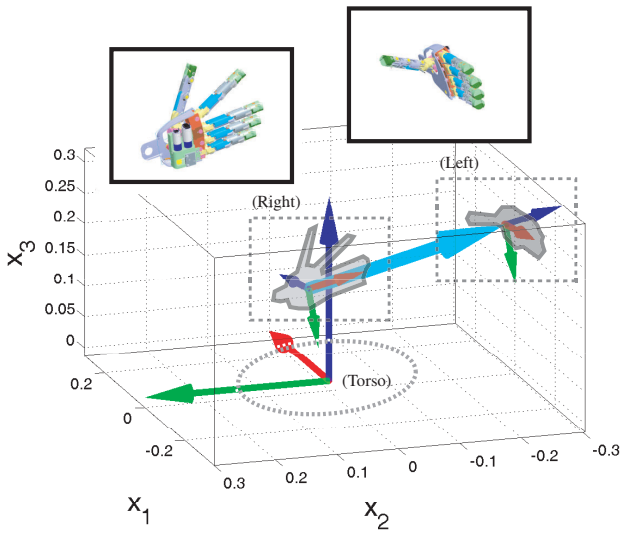


Fig. 8. Orientations for the two hands of the iCub robot during the archery task. *Top:* The right hand and the left hand CAD models of the iCub hands. *Center:* The fixed coordinate frames of reference attached to each hand are shown, as well as the main frame of reference attached to the torso of the robot. The thick blue arrow shows the relative position of the two hands which is controlled by the learning algorithm during the learning sessions. The current configuration corresponds to the robot's posture in Fig. 7.

to consider a desired rest position without having to define an explicit hierarchy as in a standard nullspace formulation. The optimization is defined as:

$$q_o = \arg\min_q \Big(w_1 \big(\hat{n} - n(q)\big)^\top \Sigma_n \big(\hat{n} - n(q)\big) + w_2 \big(\hat{q} - q\big)^\top \Sigma_q \big(\hat{q} - q\big) \Big), \quad (9)$$

$$\text{u.c.} \quad \left| \begin{array}{l} \big(\hat{x} - x(q)\big)^\top \Sigma_x \big(\hat{x} - x(q)\big) < \epsilon, \\ q > q_{\min}, \\ q < q_{\max}. \end{array} \right.$$

In the above equation, $q_o, q \in \mathbb{R}^6$ are joint angles within bounds $q_{\min}$ and $q_{\max}$. $x(q) \in \mathbb{R}^3$ and $n(q) \in \mathbb{R}^3$ are respectively the position and orientation of the end-effector (the orientation is represented as an axis-angle representation). Σ_q, Σ_x and Σ_n are covariance matrices used to modulate the influence of the different variables. In the experiment presented here, the use of identity matrices was sufficient to obtain natural looking motions. ϵ is a predefined error value.

The posture of the iCub's arms and the grasping configuration for the bow and the arrow are shown in Fig. 1. During the experiment, while the robot is learning, between every trial shot it is adjusting the relative position and orientation of the two hands, which in turn controls the direction and speed of the arrow. The desired relative position between the two hands is given by the learning algorithm before each trial. The desired orientation of the two hands is calculated in such a way, so that the bow is kept vertical (i.e. zero roll angle). The left hand's palm is kept perpendicular to the desired arrow direction, while the right hand's fingers are

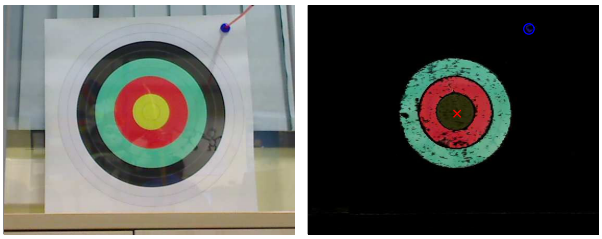


Fig. 9. Detection of the target and the arrow. *Left:* The camera image. *Right:* The pixels are filtered based on their likelihood of belonging to the target model or the arrow model. The red cross indicates the estimated center of the target. The blue circle indicates the estimated position of the arrow.

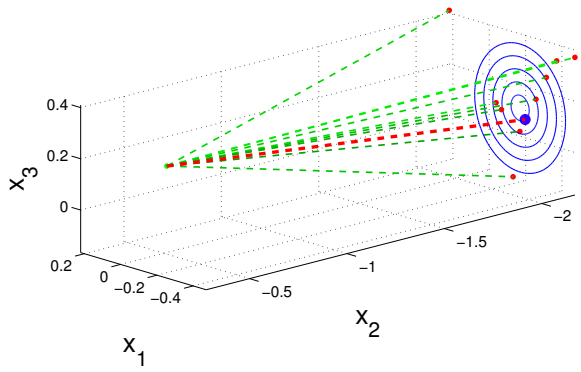


Fig. 10. Results from a real-world experiment with the iCub robot. The arrow trajectories are depicted as straight dashed lines, because we do not record the actual trajectories from the real-world experiment, only the final position of the arrow on the target's plane. In this session the ARCHER algorithm needed 10 rollouts to converge to the innermost ring of the target.

kept aligned with the arrow direction. Fig. 8 illustrates the orientations for the two hands.

C. Experimental results

The real-world experiment was conducted using the proposed ARCHER algorithm and the proposed image processing method. The camera was attached to the metal frame holding the robot and produced images with a resolution of 1280×720 pixels. An example detection of the target and the arrow is shown in Fig. 9.

For the learning part, the number of rollouts until convergence in the real world is higher than the numbers in the simulated experiment. This is caused by the high level of noise (e.g. physical bow variability, measurement uncertainties, robot control errors, etc.). Fig. 10 visualizes the results of a learning session performed with the real robot. In this session, the ARCHER algorithm needed 10 rollouts to converge to the center.

A sequence of video frames showing the learned real-world arrow shooting is shown in Fig. 11. The video is available online at: http://programming-by-demonstration.org/videos/humanoids2010/.

VI. DISCUSSION

For the archery task, the original idea was to teach the robot to pull the arrow by itself before releasing it, but this turned out to be too difficult, because of the quite limited

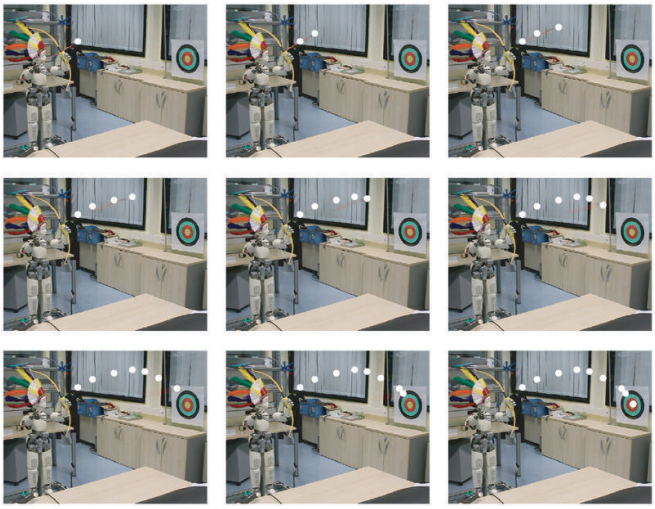


Fig. 11. Sequence of video frames showing the learned real-world arrow shooting. The arrow hits the central yellow part of the target.

range of motion and range of orientation of the two arms. The narrow joint angle ranges makes the workspace very limited and prohibits performing human-like movement especially in horizontal direction. Thus, some aspects of the bi-manual coordination were hard-coded to bypass these hardware limitations. We had to modify the shooting position by folding both arms in order to achieve maximum maneuverability for the two hands. Also, we simplified the procedure for hooking the arrow to the string, because of the difficulty in grasping, pulling and releasing the rope.

With a RL algorithm, it is possible to incorporate a bias/preference in the reward. For ARCHER, a similar effect could be achieved using a regularizer in the regression.

In the future, we plan to extend the proposed method by adding imitation learning in order to teach the robot how to perform the whole movement for grasping and pulling the arrow.

VII. CONCLUSION

We have presented an integrated solution which allows a humanoid robot to shoot arrows with a bow and learn on its own how to hit the center of the target. We have proposed a local regression algorithm called ARCHER for learning this particular skill, and we have compared it against the state-of-the-art PoWER algorithm. The simulation experiments show significant improvement in terms of speed of convergence of the proposed learning algorithm, which is due to the use of a multi-dimensional reward and prior knowledge about the optimum reward that one can reach. We have proposed a method for extracting the task-specific visual information from the image, relying on color segmentation and using a probabilistic framework to model the objects. The conducted experiments on the physical iCub robot confirm the feasibility of the proposed integrated solution.

VIII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the help of Ugo Pattacini for the inverse kinematics controller of iCub, the support of Dr. Vadim Tikhanoff for running the iCub simulator, and the invaluable help of Prof. Darwin G. Caldwell for improving the quality of this manuscript.

REFERENCES

[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Secaucus, NJ, USA: Springer, 2008, pp. 1371–1394.

[2] B. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, 2009.

[3] R. Sutton and A. Barto, *Reinforcement learning: an introduction*. Cambridge, MA, USA: MIT Press, 1998.

[4] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 229–256, 1992.

[5] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomput.*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[6] J.Peters and S.Schaal, "Policy gradient methods for robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems IROS*, 2006.

[7] J. Kober and J. Peters, "Learning motor primitives for robotics," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, May 2009, pp. 2112 –2118.

[8] J. Kober, "Reinforcement learning for motor primitives," Master's thesis, University of Stuttgart, Germany, August 2008.

[9] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot motor skill coordination with EM-based reinforcement learning," in *The 2010 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS2010)*, 2010.

[10] Wikipedia. (2010) History of archery; Kyudo. [Online]. Available: http://en.wikipedia.org/wiki/History_of_archery; http://en.wikipedia.org/wiki/Kyudo

[11] Y. Suematsu. (2001) Zashiki karakuri. Department of Electronic-Mechanical Engineering, Nagoya University. [Online]. Available: http://www.karakuri.info/zashiki/index.html

[12] E. Gribovskaya and A. Billard, "Combining dynamical systems control and programming by demonstration for teaching discrete bimanual coordination tasks to a humanoid robot," in *Proc. ACM/IEEE Intl Conf. on Human-Robot Interaction (HRI)*, 2008.

[13] G. Schwarz, "Estimating the dimension of a model," *Annals of Statistics*, vol. 6, pp. 461–464, 1978.

[14] G. Metta, G. Sandini, D. Vernon, L. Natale, and N. F., "The icub humanoid robot: an open platform for research in embodied cognition," in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, Washington DC, USA, 2008, pp. 50–56.

[15] N. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A. Ijspeert, M. Carrozza, and D. Caldwell, "iCub: The design and realization of an open humanoid platform for cognitive and neuroscience research," *Advanced Robotics*, vol. 21, no. 10, pp. 1151–1175, 2007.

[16] S. Lallee, S. Lemaignan, A. Lenz, C. Melhuish, L. Natale, S. Skachek, T. v. D. Zant, F. Warneken, and D. P. Ford, "Towards a platform-independent cooperative human-robot interaction system: I. perception," in *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2010)*, 2010.

[17] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: Yet another robot platform," *International Journal on Advanced Robotics Systems*, vol. 3, no. 1, pp. 43–48, 2006.

[18] P. Fitzpatrick, G. Metta, and L. Natale, "Towards long-lived robot genes," *Robotics and Autonomous Systems*, vol. 56, no. 1, pp. 29–45, 2008.

[19] U. Pattacini, F. Nori, L. Natale, G. Metta, and G. Sandini, "An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots," in *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2010)*, 2010.