Learning from Failed Demonstrations in Unreliable Systems

Akshara Rai, Guillaume de Chambrier and Aude Billard

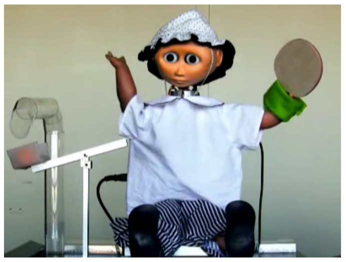
***Abstract*— This paper presents a method to teach a robot to play Ping Pong from failed demonstrations in a highly noisy and uncertain setting. To infer useful information from failed demonstrations, we use a MultiDonut Algorithm [7] that minimises the probability of repeating a failed demonstration and generates new attempts similar but not quite the same as the demonstration. We compare human demonstrations against a random strategy and show that human demonstrations provide useful information and hence yield faster learning, especially in higher dimensions. We show that learning from observing failed attempts allows the robot to perform the task more reliably than any individual demonstrator did. We also show how this algorithm adapts to gradual deterioration in the system and increases the chances of success when interacting with an unreliable system.**

I. INTRODUCTION

The ability of an agent to learn a control policy to accomplish a specific task given partially correct information provided by a teacher (human or another agent) is a vital component for successful transfer of knowledge. Assuming that the teacher is an expert in the specific task at hand is restrictive. Ball games are a specific example, where quite often the teacher knows the goal of the game but might very well be suboptimal in demonstrating it. In other words it is necessary for the student/agent to learn from failure and self-improve, to be able to quickly attain the goal. This work takes a *Programming by Demonstration* (PbD) approach to learn a control policy in which the training data provided by human teachers is mostly suboptimal.

We consider a task in which a human teaches a robot to play a Ping-Pong like game. The set-up is similar to an arcade, where the user can control an articulated robot through a joystick (see figure 1). Through this interface, the players controlled the launch of the ball, with the right arm of the robot, and the hitting motion, with the left arm. A set of volunteers were asked to play the game during which their control actions were recorded. For each set of demonstrations a novice user is asked to play repeatedly for a fixed number of times. This way we leave enough time for each player to learn how to play the game. As they do so, we gather numerous failed, as well as successful trials, which we can then use for training our robotic system. The catapult which the robot must hit to launch the ball is imprecise and leads to stochastic behaviour. Hence, reproducing simply previously successful strategies leads to failure from time to time. The robot must hence learn a strategy to optimize its average successful trials and use human demonstration solely as a guidance but not as ground truth.

Teaching skills to agents by demonstrating by an experienced agent is the focus of a large area of research referred


(a)

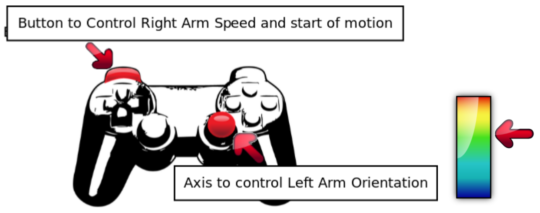

Button to Control Right Arm Speed and start of motion
Axis to control Left Arm Orientation
(b) (c)

Fig. 1. **The Ping-Pong Set Up** (a) The humanoid robot, Robota, with the ping-pong racket attached to its right arm and a catapult with a ping-pong ball.(b)The joypad used to control the orientation of left arm and speed of right arm. (c) Speed Level scaled from 0 (blue) to 1 (red) to control speed of right arm

to by the umbrella term PbD, also commonly known as *Imitation Learning*, the reader is referred to [3] and [2] for a literature survey on the field at large.

Here, we consider ways in which the robot self-improves by searching actively region around the demonstrations. This approach resembles in its principle Active Learning (AL) [5],[12] (also known as *tutelage*). AL is a sub-field in which the agent actively queries the teacher about specific aspects of the task [6], which he is uncertain about so as to increase the learning rate. In AL there is a strong focus on high level, *outcome-defined* tasks, where the goal is formulated with symbols and intent is communicated through social cues, such as natural language and facial expressions, see [4], [9]. Interactive Learning (IL) is similar to AL, with the exception that the learner doesn't query the teacher about aspects of the task but rather is continuously corrected by the teacher. IL research is focused on *process-defined* tasks, where the goal is the refinement of a skill and teaching is commonly done kinesthetically, see [13]. Both AL and IL are interactive learning procedures where multiple refinements and corrections of the control policy are performed. The drawback is that self-improvement cannot be achieved autonomously (the teacher encodes the task) and the teacher has to be an expert, which as stated previously can be disadvantageous. Reinforcement Learning (RL) refers to a wide area of techniques whereby the agent self-improves through trial and error given partially successful demonstrations. RL has been combined with imitation learning [10], [8], [11] to provide a framework where a teacher can initialize the control policy through demonstrations and then lets the student search autonomously for the correct policy. The search is driven by a reward signal provided by the teacher. The drawback is the specification of the reward function is often non-intuitive. A remedy is Inverse Reinforcement Learning (IRL) [1], where the reward signal is inferred from the demonstrations. However it requires the choice and predefinition of features, whose weighted combination encodes the reward signal, and often requires very good demonstrations.

This calls for the design of most general formulation to learn from a teacher; where the task does not need to be encoded explicitly and where the teacher can be suboptimal.

We build on a recent work whereby we learn from failed demonstrations [7]. In this work, we had proposed a stochastic search process that decreases the probability of selecting the demonstrated portions of the joint space that have high variance in strategy space whilst increasing the areas with low variance. The rational was that one should reproduce parts of the demonstrations where all demonstrators agree (as this likely mean that this encapsulates the essence of the task at hand), while avoiding the repeat regions of the trials where demonstrators disagree (as this may reflect their respective uncertainty as to how to solve the task).

While in our early work, the robotic system was deterministic, we here investigate how the approach can be used to allow the robot to adapt in the face of a stochastic robotic system whose behaviour changes with time.

A crucial aspect of our work is that the robot should be able to autonomously succeed at the game given poor training data and be able to outperform or be at the same level of an expert user. In other words, given a suboptimal starting point it should reach optimality without the help of the teacher.

A. Understanding the System

The system at hand is not the commonly encountered Ping-Pong game. First, the robot hits a catapult with its right arm, to launch the ping pong ball. The hitting speed of this arm (decided by user with help of a color-bar) determines the launching speed of the ball. The robot then hits the ball with its left arm, which holds a ping pong racket, see Figure 2. The user controls the end position of this arm, and its speed is kept constant.

During the human training stage, the robot is controlled by the human teacher. By pressing on a switch the teacher selects the speed on the color-bar and initiates the launching motion. This automatically drives the right arm to hit the catapult. A joystick allows the teacher to move the 2 degrees of freedom left arm to guide the hitting motion (see Figure 1).

Our very first observation was that the task was extremely difficult to achieve, by humans as well as the robot as it was very fast and uncertain. It took a little while before

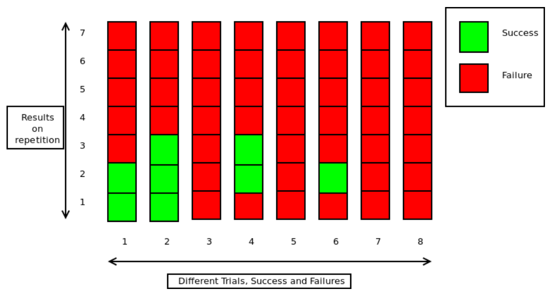


Fig. 3. **Unreliability of the system.** We repeated the same input 7 times on our set up in consecutive trials and observed their results. This was done for 8 different starting points– successes and failures. We observed that there is never a success after four consecutive iterations. Success could be obtained by starting from success as well as failed initializations. These results are due to the highly noisy and unreliable nature of our set up.

humans could get accustomed to the game and even then, they couldn't succeed in all trials. Apart from the variable path of the ball, the robot also has errors introduced from its motors accumulating over time. This makes the system highly unreliable as the robot performance deteriorates very rapidly over the experiments.

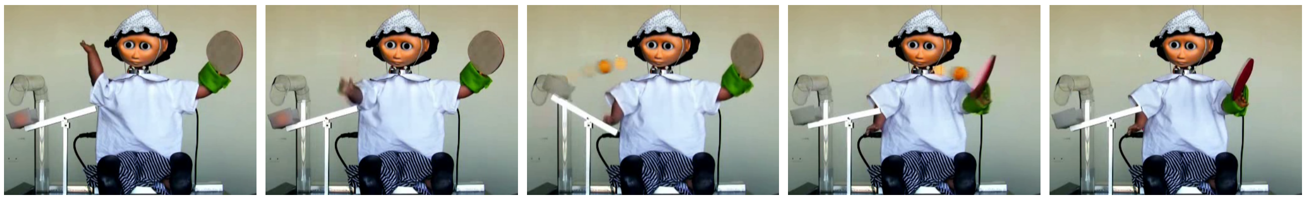
One could query why use such an unreliable platform for conducting the experiments. We see value to have a system that deteriorates over time, as this is quite realistic. All machines deteriorate after usage and humans still manage to get the best of them. Stochastic systems with imperfect responses are plenty and we learn how to deal with these to maximize our success rate over time (opening a garage door with an infra-red remote fails regularly for lack of good detection of the signal or gripping of the motors due to change in humidity and temperature). To cope with such stochastic systems, humans learn which actions are *most likely* to yield success (for instance, changing slightly the direction in which we direct the remote to better align it with the receiver which is usually invisible, or pressing several times consecutively on the remote to give more boost to the door and increase our chance to overcome friction when needed). This ability that humans have to learn the "feasibility region", that is the region of actions that yield on average success is what we seek to reproduce here.

To estimate the unreliability in our system, we repeatedly played the same game strategy 7 times and observed its output. This was repeated for 8 different input starting points– successes and failures to have a fair estimate. The results are shown in Figure 3. Along the x-axis we see different input parameters and along the y-axis, their results on repetition.

II. PROBLEM STATEMENT

A. Task at hand

In a nutshell, the game can be played by controlling three parameters: Speed of launch of the ball (with right arm): v (scaled from 0 to 1), Delay between launch of ball and strike: ΔT (ms), Joint angles for robot's left (striking) arm: $\theta = [\theta_1, \theta_2]$ (in degrees).



(a) (b) (c) (d) (e)

Fig. 2. **Sequence of actions in the game:** (a) Starting Position (b) Motion of left arm hitting the catapult- speed of hitting controlled (c)-(d) Flight of ball and motion of right arm to hit the ball (e) End of game.

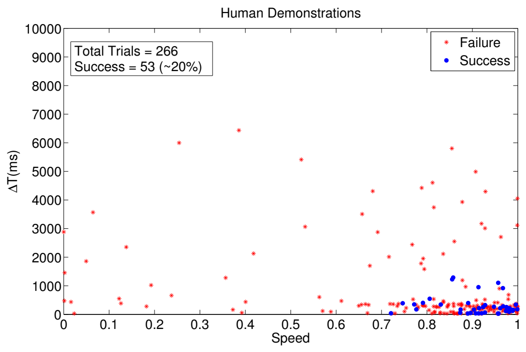


Fig. 4. **Human Trials using the Joypad.** The figure shows the human demonstrations collected over experiments. The humans could control the speed of left arm (x-axis) and the time delay between launch of ball and onset of motion of right arm (y-axis).

Parameter	Range	Mean		Variance	
		Success	Overall	Success	Overall
Speed	[0 to 1]	0.91	0.81	0.08	0.24
ΔT (ms)	[0 to ∞]	298.04	695.20	325.43	637.50
θ_1 $^\circ$	[-105 to 21.27]	0.0140	0.216	0.0083	0.113
θ_2 $^\circ$	[-105 to 21.27]	21.27	14.83	0	9.87

TABLE I

MEAN AND VARIANCES OF THE PARAMETERS OVER SUCCESSFUL TRIALS AND ALL TRIALS. PARAMETERS θ_1 AND θ_2 HAVE VERY LOW VARIANCE AS COMPARED TO SPEED AND ΔT OVER SUCCESSFUL TRIALS.

Since ΔT is typically very small, the game is quite fast (for success: $298ms \pm 322ms$) and needs the user to react very quickly. Also, only balls launched by high speeds could be hit by the right arm, making the selection of the correct speed important. To add to this, the ball could follow different paths in its flight and the robot motors had some noise, which would accumulate over time. All this made the task quite difficult for humans as well as the robot. It took a little while before humans could get accustomed to the game and even then they couldn't succeed in all trials.

B. *Experiments with Humans*

We asked 13 human subjects to play around 20 trials (sometimes more as the robot didn't respond well due to some communication error) with the robot to give us some estimate of the distribution of the success and failure region in the whole parameter space. As can be seen in Figure 4, the success region is very small as compared to the whole parameter space. A total of 266 trials led to 53 successes, concentrated in the bottom right, shown by blue dots. We see a larger spread of failed trials, shown in red.

C. *Formulation*

To understand the significance of each parameter in determining success or failure, we measure their variance over several human trials, shown in Table I.

When we compare the total range of the last two parameters with their variance in successful demonstrations, we see that success is characterised by a unique point in the space of θ_1 and θ_2. The geometry of the set-up does not allow for a collision of the ball and ping-pong racket in other configurations of the arm. This was observed easily as this point was on the edge of the workspace of the arm. Hence we performed a learning primarily over the first two parameters- the speed of hit and the ΔT.

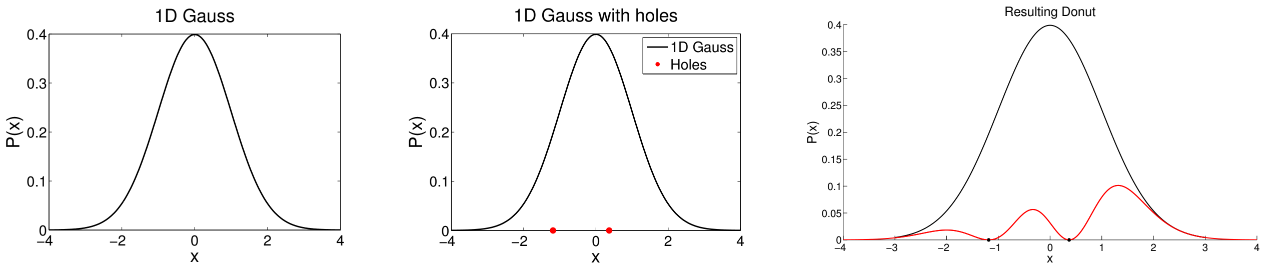
The aim of our learning was to maximize the probability of successfully striking the ball by the robot. This required learning a relation between the 'feasible' launching speed and the ΔT between the launch and the strike. Thus, we tried to learn this mapping using a stochastic search algorithm in the space of the two variables, as explained next.

D. *Stochastic Search Model*

We need a model that can search for 'good' strategies in the space of the two variables using information provided by an initial demonstration. In particular, we would like our search algorithm to have the following three qualities:

- Reduce the probability of repeating a failed strategy
- Assuming the initial demonstration was close to a good demonstration, increase probability of the area around the demonstration
- Favouring one set of actions over others by increasing, or decreasing, the likelihood to pick these must be done as a function of how good or bad these were.

Such a distribution was introduced in [7] using combinations of Gaussians, known as the Donut distribution. The Donut is a difference between two Gaussians, centred over two different points giving rise to a off-centre distribution as shown in Figure 5. The next strategy is the most likely strategy in the resultant distribution. This suits our situation as now we move away from failed demonstrations by reducing the probability of a failed trial. Also, we remain close to initial demonstrations by increasing probability of regions around it and explore them, leading to an overall shift in our strategies in the parameter space.



(a) (b) (c)

Fig. 5. **Illustration of a Donut on a 1D Gaussian.** The black curve shows the original Gauss and the red curve is the Donut distribution (a) 1D Gauss. (b) Failed Demonstrations - 'Holes' represented by red dots on the x-axis. (c) Resultant Donut with minima at holes and higher probability in region around the hole. Note that the holes outside the 3σ range of the Gaussian do not affect the Donut Distribution.

To perform learning over more than one demonstration we need to combine more number of Donuts. We use, instead of k distributions with k holes, just one distribution with k holes as formulated below.

$$P(\mathbf{x})=\frac{1}{Z}\mathcal{N}(\mathbf{x}|\mu^0,\Sigma^0).\prod_{k=1}^{K}(1-exp(-\frac{1}{2}(\mathbf{x}-\mu^k)^T\Sigma^{k^{-1}}(\mathbf{x}-\mu^k)))$$

The naught distribution is created over initial human demonstrations and keeps the MultiDonut close to human data. The multiplied distributions here take the place of individual Donuts and are centred over the failed attempts, called 'holes'. The height that the resultant MultiDonut assumes at a hole and the distance of the maxima or the next most likely strategy from this point can be tuned by setting a weight to each hole. Z is a normalising constant to keep integral of the distribution equal to 1. We use the MultiDonut distribution for all our experiments henceforth.

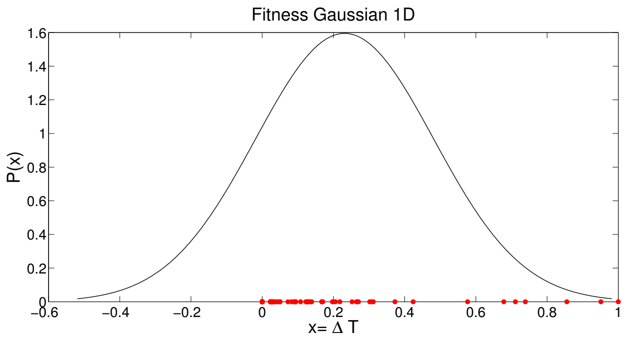
E. Sampling

Since this distribution is always multi-modal, we cannot use gradient methods for finding the maxima. Thus, we use a sampling method based on rejection sampling to draw a sample from the Multidonut distribution, and use it as the next trial parameters. This also introduces some uncertainty and randomness in our search resulting in a higher exploration of the parameter space.

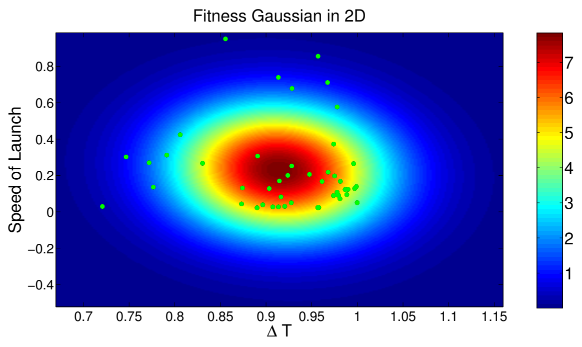
F. Fitness Function - The Fitness Gaussian

The result of each trial was binary- 1 if we hit the ball and 0 if we failed. However, to determine a continuous fitness of our trials, we modelled the fitness function by fitting a Gaussian on a set of successful human trials gathered earlier. In the initial iterations, this was used to classify a trial as success or failure to do a quick, noise-free search for successful trials using the MultiDonut. Also, the distance from the mean gave the MultiDonut an estimate of how good or bad the trial was, which was used in setting the height and width of our holes.

We considered a 1D Gaussian (using only ΔT) and 2D Gaussian (using ΔT and speed) instead of a GMM as the success points should be cluttered together in the space of the parameters. The respective Gaussians can be seen in Figure 6.



(a)



(b)

Fig. 6. **Gaussian models:** These Gaussians give the probability of winning a game given a particular set of parameters. Note that the ΔT was scaled from 0 to 1 to model the Gaussian. The success points should be cluttered together but are not due to false negatives by the system. Thus, to model a noise free system, it is better to have a single Gaussian over the success points. (a) **1D:** The success points are shown as the red dots on the x-axis. Note that the time was scaled from 0 to 1 to model the Gaussian.(b) **2D:** The red regions represent regions of high probability and the blue are regions of lower probability. The green dots are the success points over which the Gaussian was drawn.

G. Algorithm

The fitness of our trials was measured using the Euclidean distance of these points in the parameter space from the mean of our Fitness Gaussian. These were classified into success or failures using a threshold. The points lying outside the threshold region (line segment in 1D and circle in 2D) were classified as failures while those inside as successes. This fitness measure was also used to set the height and width of the holes by modulating their prior weights. It also modulated the weights of the successful demonstrations used to form the positive distribution.

The search algorithm using MultiDonut distribution is shown in Algorithm 1.

Algorithm 1: The MultiDonut algorithm used

```
Input: Ping-Pong Trials- Random Initializations or
       Demonstrations
Output: New set of parameters
while $TotalSuccess \leq N$ do
    foreach Trial $i$ do
        Calculate Fitness($i$);
        if Fitness(i) $\geq$ Threshold then
            $Success = 1$;
            $TotalSuccess++$;
        else
            $Success = 0$;
    if $TotalSuccess == 0$ then
        Draw Gaussian over Failed initializations;
    if $0 < TotalSuccess < 10$ then
        Draw GMM over successful initializations with
        $\Sigma^0 \propto \Sigma^{all}$;
    if $TotalSuccess > 10$ then
        Fit GMM over successful initializations;
    foreach Failed Demonstration $j$ do
        Create holes in the GMM using Fitness
        measure;
        Sample resultant Multidonut to get the new
        most likely point in parameter space;
```

There are two major stages in our MultiDonut learning algorithm– when it has only failed demonstrations to learn over and when it has some success and some failed demonstrations. These are explained in details in the following.

1) Learning only from Failed Demonstrations: When the MultiDonut has only failure points to learn over, the positive distribution was a Gaussian fitted over this data. The negative distribution was centred on the holes and the height and width of these holes was determined using the distance of the corresponding centre from the mean of the Fitness Gaussian.

2) Learning from Successful and Failed Demonstrations: If the MultiDount has as input some success points along with failures the positive distribution is built over these successful points and the negative distribution remains the same as before. This means that after finding a successful strategy, Multidonut exploration limits to points in a smaller space around the successful points.

III. Results

In this section, we introduce two metrics we used for evaluating the performance of the MultiDonut. We then compare the performance of humans with our MultiDonut.

To show that initializing the MultiDonut with human demonstrations improves the performance, we contrast it with the MultiDonut search when initialized with random trials. We also compare the performance of the MultiDonut in 1D (i.e. using a distribution on ΔT) versus in 2D (i.e. both ΔT and v). We finally show the results of implementing the MultiDonut on an unreliable system and the subsequent improvement in performance.

We evaluate the performance of the MultiDonut based on two metrics given below:

1) **Measure 1**: Number of iterations needed to reach 10 successful trials
2) **Measure 2**: Success rate of Multidonut after the first successful trial

For a graphical description of the metrics refer to the Figure 7.

The first measure is an estimate of how long the Multidonut takes to search the region of high success probability. The second measure gives us an estimate of the consistency of the Multidonut after reaching the first success.

After reaching 10 successes, the positive distribution is drawn over these new success points using their variance as the variance of the distribution. Thus, the percentage of success after reaching 10 successes is very high $90\% - 100\%$ as the MultiDonut distribution almost duplicates the fitness Gaussian.

The MultiDonut was initialized using two different initializations, as described below.

- *Random Initialization:* 5 random points were given as initial input to the MultiDonut
- *Human Demonstrations:* 5 human demonstrations given as initial input to the MultiDonut

The random initializations or Human Demonstrations could be success or failures, though starting with a successful point makes the learning of the MultiDonut very quick. With random initializations, this was more probable in 1D as they could explore a large portion of the parameter space, compared to in 2D. Thus, while intuitively we would expect Human Demonstrations to be better initialization methods than Random, as they contain useful information about the parameters, it is more clearly evident in 2D. In 1D, human initialization and random initializations perform competently, but in 2D we see that human demonstrations excel over random initializations.

Thus, we can say that in 2 dimensions human demonstrations, even if failed, are good initialization points for a learning method. We can then extrapolate this idea to higher dimensions where the space to explore becomes even larger

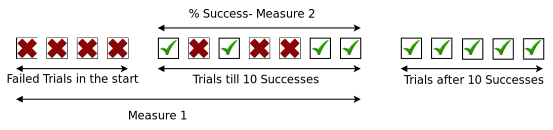
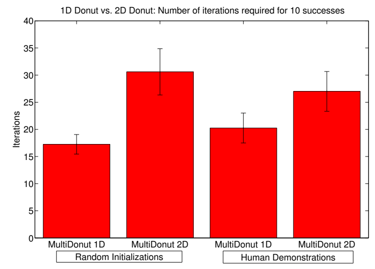
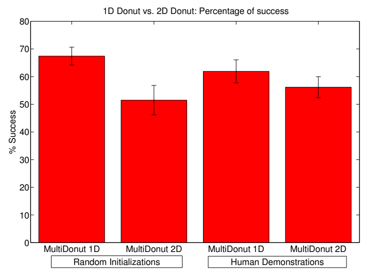


Fig. 7. **MultiDonut Metrics**- The blocks represent consecutive trials. The failures are shown with red crosses while successes with green ticks. The first block are the trials with only failures at the beginning of the experiment, when the MultiDonut is learning. The second block starts with the first success and continues until 10 successes. The third block represents trials after 10 successes till 20 successes.

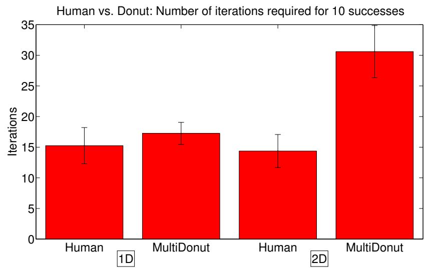


(a)

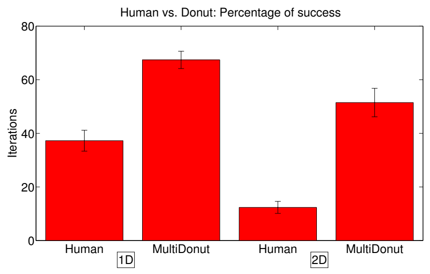


(b)

Fig. 8. Comparison of mean and variance of Multidonut performance for human and random initialization. (a) **Measure 1**- Iteration for 10 successes (b) **Measure 2**- % Success after 1 success



(a)



(b)

Fig. 9. Comparison of mean and variance of Multidonut performance and humans for 1D and 2D. (a) **Measure 1**- Iteration for 10 successes (b) **Measure 2**- % Success after first successful trial

and good human demonstrations are capable of starting the MultiDonut from good regions closer to success.

When we compare the performances of the human subjects with those of the MultiDonuts (Figure 9), we observe that even though humans reach success faster than the MultiDonut, they are unable to reproduce that success with certainty. Thus, the iterations needed to reach the first successful trial are smaller and so is the percentage of success after reaching 1 success. The MultiDonut on the other hand might take a longer time to reach the first successful trial but performs quite consistently after reaching it.

Not all human demonstrators could produce 10 successful trials in 20 trials, as calculated using the fitness Gaussian. So we included only those who managed to produce 10 successes to compare with the MultiDonut. Thus, this is a comparison between the expert humans and the MultiDonut. We see that the humans are much quicker and take almost equal number of trials in 1D and 2D to obtain 10 successful trials while the number of trials almost doubles for the MultiDonut.

IV. Implementing on the Unreliable System

In Section I-A we had shown how the system was found to be highly unreliable and biased towards failure and any successful set of parameters could not produce more than about 4 successes on repetition. This was explained by the errors in the motors of the robot arm, which caused the launching speed of the ball to decrease, making an earlier successful strategy a failure.

We here investigate how these successive failures can be used by the Multidonut system to update its strategy, so as to adapt to the deterioration of the robotic system on-line. A MultiDonut can update its strategy by incorporating the result of each trial, whether a success or a failure. If a previously successful point becomes a failure, the MultiDonut inserts a hole (albeit small) at this point. This reduces the probability that this strategy (i.e., set of parameters) be selected again, while increasing the likelihood of the neighbouring regions. This leads the MultiDonut to sample away from this region, which now represents failure, and to explore other points in its neighbourhood.

To test whether this on-line update of the MultiDonut did indeed increase the rate of success, we proceeded as follows. We first trained our MultiDonut on the Fitness Gaussian until it generated a set of parameters that corresponded to a successful trial (as per the *Fitness Gaussian*). This set of parameters was then run on the robot system and the outcome of the trial was used to update the MultiDonut distribution. The method for setting the height and width of the hole was identical to that used in the first set of experiments, i.e. it was set proportional to the euclidean distance of the point from the mean of the Fitness Gaussian, see Section II-F.

We compared the ‘success rate’ of this MultiDonut to a simple repetition of a successful trial (on the robot). As mentioned earlier, repeating the same strategy stopped being successful after around 4 repetitions due to errors in the motors. On the other hand, MultiDonut was able to score 5-8 successes in 10 trials by exploring neighbouring regions of a now failure strategy and updating its strategy. However, we were able to implement this only in 1D and it would be interesting to see how this behaves in 2D where the region to explore is larger.

To explain the above concept better, we show the progress of the trained 1D MultiDonut on the robot in one typical trial alongside a trial in which an initially successful strategy was repeated several times in Figure 10.

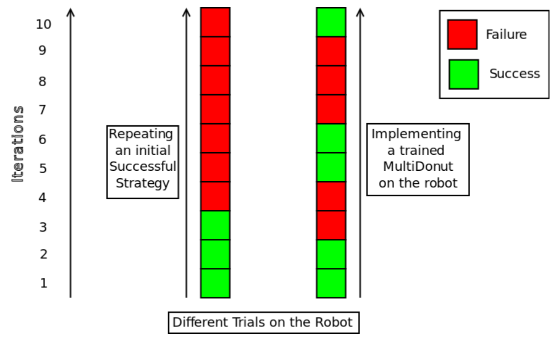


Fig. 10. Results on repeating an initially successful strategy and implementing the trained MultiDonut on a robot. The green boxes are successful outcomes while the red ones show the failures. When simply repeating a strategy, no success is obtained after three iterations. The MultiDonut adapts to the system bias and changes the strategy.

We implemented a trained MultiDonut starting from random initializations and human demonstrations in 1D on the robot and found them to be producing very similar success rates. Both were found to score on an average 6 successes out on 10, over 5 different trials for each. This is because, once trained, the MultiDonut behaves similarly for both types of initializations. Also, our implementation is in 1D, for which both the types of initializations are comparable.

V. CONCLUSIONS

In this paper we presented a method to teach a robot from failed demonstrations in an unreliable system. The MultiDonut algorithm described can eliminate regions of parameters that lead to failure and remains in the region that leads to success once it has reached this region. It also helps to modify an earlier good strategy to adapt to changes in the changing system dynamics.

We also compared using human demonstrations to random initializations to train our MultiDonut. Both perform comparably for lower dimensions but for higher dimensions, human demonstrations perform better. This is expected as for lower dimensions random initializations are able to explore a large part of the space, but with increasing dimensions human intuition comes in useful by driving the system closer to good starting regions with higher probability of success.

The convergence for a MultiDonut is faster for 1D and more reliable than for 2D. This raises concerns over the convergence of the method for higher dimensions. But since the method is actually very dependent on the type of initialization in high dimensions, a good starting demonstration even in high dimensions might lead to quick convergence of the MultiDonut.

VI. ACKNOWLEDGEMENTS

This work was funded in part by the European Commission under contract number FP7-248258 (First-MM).

REFERENCES

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *In Proceedings of the Twenty-first International Conference on Machine Learning*. ACM Press, 2004.

[2] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483, May 2009.

[3] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. *Handbook of robotics*, 1, 2008.

[4] C. Breazeal. Tutelage and collaboration for humanoid robots. *International Journal of Humanoid Robotics*, 01(02):315–348, 2004.

[5] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models, 1995.

[6] J. de Greeff, F. Delaunay, and T. Belpaeme. Active robot learning with human tutelage. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pages 1–6, 2012.

[7] Daniel H. Grollman and Aude G. Billard. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 4:331–342, 2012.

[8] Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Auton. Robots*, 34(1-2):111–127, 2013.

[9] A. Lockerd and C. Breazeal. Tutelage and socially guided robot learning. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 4, pages 3475–3480 vol.4, 2004.

[10] Jan Peters and Jens Kober. Using reward-weighted imitation for robot reinforcement learning. *Adaptive Dynamic Programming and Reinforcement Learning*, pages 226–232, 2009.

[11] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research - Proceedings Track*, 15:627–635, 2011.

[12] David Silver, J. Andrew (Drew) Bagnell, and Anthony (Tony) Stentz. Active learning from demonstration for robust autonomous navigation. In *IEEE Conference on Robotics and Automation*, May 2012.

[13] Aude Billard Sylvain Calinon. Incremental learning of gestures by imitation in a humanoid robot. In *In Proceedings of the 2007 ACM/IEEE International Conference on Human-Robot Interaction*, pages 255–262, 2007.