Preprint 22.6.2014 - To appear in Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014) - (c) IEEE

Learning of Closed-Loop Motion Control

Farbod Farshidian, Michael Neunert and Jonas Buchli

***Abstract*— Learning motion control as a unified process of designing the reference trajectory and the controller is one of the most challenging problems in robotics. The complexity of the problem prevents most of the existing optimization algorithms from giving satisfactory results. While model-based algorithms like iterative linear-quadratic-Gaussian (iLQG) can be used to design a suitable controller for the motion control, their performance is strongly limited by the model accuracy. An inaccurate model may lead to degraded performance of the controller on the physical system. Although using machine learning approaches to learn the motion control on real systems have been proven to be effective, their performance depends on good initialization. To address these issues, this paper introduces a two-step algorithm which combines the proven performance of a model-based controller with a model-free method for compensating for model inaccuracy. The first step optimizes the problem using iLQG. Then, in the second step this controller is used to initialize the policy for our PI^2-01 reinforcement learning algorithm. This algorithm is a derivation of the PI^2 algorithm enabling more stable and faster convergence. The performance of this method is demonstrated both in simulation and experimental results.**

I. INTRODUCTION

Motion planning and control is one of the long standing challenges for the robotics community. A common approach is to separate the motion trajectory design from the controller design. Usually, a planner generates an open loop motion and a feedback tracking controller then tries to force the system to generate this desired motion. As both elements are designed separately and might even have competing goals, their combination is often suboptimal. In contrast, a combined design approach allows one to optimize a single controller that encodes both the motion planning and the tracking behavior at the same time. Optimizing this controller will directly lead to a better overall performance because the controlled system dynamics can be better leveraged to generate smoother and more agile motion.

Using control theory, the combined motion planning and control problem can be transformed into an optimization problem where the dynamics are represented as hard constraints. The result of this optimization is a closed loop controller which contains a feedforward and a time-dependent feedback term. Existing methods to solve this problem can be grouped into model-based and model-free algorithms.

A. Model-based Algorithms

The optimal control method is one of the model-based approaches which combines the trajectory and controller design into a single problem. The iLQG algorithm [1] is one of the successful optimal control approaches which is in fact the stochastic version of the well known Differential Dynamic Programming (DDP) method [2]. In general, model-based algorithms like iLQG show superior performance in simulation, but their performance in physical applications is limited by the accuracy of the system model. Although part of this modelling inaccuracy can be prevented by taking more meticulous modelling techniques, part of it might intentionally be plugged in, either to get a simpler model or to obtain a linearizable one. These limitations restrict the application of model-based methods on physical robots.

B. Model-free Algorithms

Another approach to simultaneously finding reference trajectories and controllers is to use model-free machine learning techniques (e.g. [3], [4]). Among different approaches in this area, reinforcement learning algorithms have shown good performance both in simulation and in real world applications. One of these reinforcement learning algorithms is the Path Integral Policy Improvement (PI^2) method [5]. The policy in the PI^2 algorithm is defined as a dynamic system, namely as Dynamic Movement Primitives (DMPs), which represent a linearly parametrized policy. The policy is primarily used to learn a reference trajectory. This approach is extended in [4] to not only learn the reference trajectory, but also a time varying gain scheduling for a PD controller. Therefore, the learning algorithm tries to optimize the trajectory and the controller simultaneously. Another powerful reinforcement learning algorithm is PI-BB [6]. This algorithm is the black box variant of PI^2 where instead of using the cost function signal in each time step, only the accumulated cost of the trajectory is used.

While existing learning algorithms show good performance in refining given parameters, they have two main issues. First, their success highly depends on the initial guess because they are all local methods. Second, they just partially leverage the domain knowledge of the designer e.g. the knowledge about the system dynamics is totally ignored and just the information provided from the samples is used.

C. Combining Model-free and Model-based Algorithms

In this paper, we attempt to combine the benefits of both model-based and model-free approaches. Therefore, we propose to separate the problem into two stages: derive a model-based optimal controller and subsequently adapt this controller using a learning algorithm. In this approach the domain knowledge of the designer in the form of the system model is exploited and delivered to the learning algorithm as

Farbod Farshidian, Michael Neunert and Jonas Buchli are with the Agile & Dexterous Robotics Lab at the Institute of Robotics and Intelligent Systems, ETH Zurich, Switzerland. {farbodf, neunertm, buchlij}@ethz.ch

an optimized initial guess for the parameters. This domain knowledge can vary from a simple up to a sophisticated model. In contrast to a random initial guess, the optimized controller can already stabilize an unstable system, enabling the learning algorithm to sample useful data.

For the initial derivation of the controller iLQG is used. For the subsequent learning, we propose a new algorithm PI^2-01, which is based on PI^2. While the original PI^2 leads to unstable learning in our experiments, the modified time-averaging in PI^2-01 improves the stability. Also, the new algorithm allows for using general, linearly parametrized policies. Furthermore, the sampling policy is altered to encourage a greedier exploration strategy. Note that using a policy improvement method like PI^2-01 is a design choice. One can also consider to refine the available model through hardware experiments and then use a model-based approach to optimize the control policy based on the updated model [7]. Although these approaches generalize better to other tasks than policy search approaches, they tend to require more samples to reach a similar performance.

To assess the performance of the proposed algorithm we are using the ball-balancing robot ("ballbot") Rezero [8]. For our tests we will use both a simulation and the real robot.

D. Motion Planning for Ballbots

Ballbots are essentially 3D inverted pendulums and hence are statically unstable, under-actuated, non-minimal phase systems. Due to this instability, control and motion planning have to be tackled simultaneously and can even have conflicting goals.

Prior to Rezero successful development of ballbots were conducted by Hollis et al. [9] and Kumagai et al. [10]. However, motion planning on ballbots has not yet been thoroughly addressed. Nagarajan et al. [11] use a separated trajectory planning and tracking controller to generate a set of motion primitives which are later connected to one trajectory by a graph search algorithm. In another approach Shomin et al. [12] suggest a differentially flat representation of the system dynamics to generate trajectories.

II. PROBLEM DEFINITION AND APPROACH

Learning motion control is a complicated task which requires learning both a feedforward and a state feedback controller. This problem can be defined as an optimal control problem where the desired behavior of the motion controller is encoded in a cost function. In order to solve this optimization problem, we approach the problem by leveraging both, information obtained from both, the approximate model knowledge as well as from sampling the real system.

A. Problem Definition

In its most general form we are assuming a control-affine non-linear system given by

$$\dot{x} = f(x) + g(x)(u + \epsilon) \tag{1}$$

where $f(x)$ denotes the transition matrix and $g(x)$ denotes the control gain matrix. The current state and control input are represented by x and u respectively. We assume that the system can be linearized i.e. $f(x)$ and $g(x)$ are differentiable. Furthermore, the system dynamics are subject to zero-mean Gaussian noise ϵ acting on the control inputs.

The motion planning problem is a finite-horizon optimal control problem. The goal is to find a linear, time-variant state feedback controller with feedforward terms by solving the optimal control problem. The controller is given by

$$u(x,t) = \mathcal{K}(t)^T x'(t) \tag{2}$$

where $\mathcal{K}(t)$ denotes a time-varying gain and $x'(t) = [1 \;\; x(t)]^T$ defines the augmented state vector.

The controller will be optimized to fulfill a certain task or behavior within a certain time horizon. In motion planning and control applications the optimization goal is to bring the system from an initial state $x(0)$ randomly drawn from a given set of initial states to a goal state $x(t_f)$ while optimizing the behavior during this transition. This task is encoded in the cost function of the optimization problem. The cost function is defined as:

$$J = E\left[h\left(x(t_f)\right) + \sum_{t=0}^{t_f-1} l\left(x(t), u(t)\right)\right] \tag{3}$$

containing the sum of intermediate costs $l(x(t), u(t))$ and a final cost $h(x(t_f))$. The intermediate cost is a function of state and control input and is calculated at each time step of the task execution except for the last step. The termination cost only depends on the state vector at the final time step.

B. Proposed Approach

In order to leverage the model knowledge, we have chosen to use the iLQG method. Depending on the accuracy of the available model, the obtained solution could provide near optimal performance on the real system.

As the system model differs from the true system dynamics, real system information is leveraged afterwards by applying reinforcement learning. For this, we are using a derivative of PI^2, namely PI^2-01 (see Section IV).

The two sequential methods are integrated by initializing PI^2-01 with the iLQG controller. As PI^2-01 shares the same assumptions for the cost function as iLQG, we can use a single cost function for both algorithms. As a result, both methods optimize towards the same target behavior. Furthermore, a shared cost function guarantees that this function is not tuned towards a specific algorithm but encodes the desired behavior in a general form.

By optimizing over a common cost function, the learning algorithm is initialized with a near-optimal policy. This is especially important as reinforcement learning algorithms are local methods and hence require a good initialization.

III. INITIALIZATION USING ILQG

As mentioned earlier, we are using iLGQ to derive an initial control policy. This algorithm provides a time indexed state feedback plus feedforward control law as indicated in equation (2). The main idea behind the iLQG algorithm is to iteratively linearize the system equations along nominal

trajectories in the states and the control inputs. Then, a LQG controller for the linearized model is computed. Finally, the system is simulated using the designed controller to get new nominal trajectories. This process continues until the solution reaches a local minimum. For more details about the iLQG algorithm we refer to [1].

Since iLQG is a local optimization method, we need to initialize this algorithm from a stable nominal trajectory. Here, we are initializing the iLQG algorithm with an infinite-time LQR controller. This LQR controller is designed for the system linearized around one of its equilibrium points and uses simple diagonal state and control cost matrices.

IV. LEARNING

A. Learning Policy

The number of parameters in the iLQG controller is proportional to the time horizon and inversely proportional to the sampling time. Per time step, the iLQG controller consists of $n+1$ parameters per control input (one for the feedforward term and n parameters for the n states). Optimizing over all of these parameters would require a lot of samples, leading to a slow learning rate. To reduce the number of parameters, we consider using smooth base functions instead of using functions indexed over time. For the present algorithm, the learning policy's base function $\Psi(t,x)$ is defined over time t and state x using Gaussian functions as follows:

$$\Psi(t,x) = \left[\Psi_i(t,x)\right]_{N\times 1} = \left[e^{-\frac{1}{2}\frac{(t-c_i)^2}{\sigma_i^2}} x'\right]_{N\times 1} \tag{4}$$

where N is the number of the total Gaussian functions and c_i and σ_i^2 are respectively the center and the covariance of the of the ith Gaussian function. As x' denotes the augmented state of size $n+1$, $\Psi(t,x)$ is a vector with length $N(n+1)$.

The center and covariance of the base functions need to be chosen according to the given problem. For simplicity, one can just distribute the Gaussian functions uniformly over the execution time interval and choose one covariance for all the functions. Also, in defining the covariance, one can use the rule of thumb that neighbouring Gaussian base functions should intersect at one third of their peak value. N is chosen proportional to the task's duration. Here, 1 to 3 bases function per second seem reasonable.

Given this base function, the policy (or in other words the controller) will be the inner product of the base function vector with the parameter matrix as follows:

$$u(t,x) = \Theta^T \Psi(t,x) \tag{5}$$

where Θ denotes the parameter matrix in which each column corresponds to one control input.

Since the controller resulting from the iLQG algorithm is used as the initial policy for the learning algorithm, it should be mapped to the policy representation as given in (5). As the number of parameters in the iLGQ parametrization is higher than the one of the learning policy, this is a many-to-one mapping. Therefore, in order to minimize the differences between both representations, least squares over equating-coefficients can be used. This method minimizes the sum of squared errors between both representations over the task's time interval. In this way, the best fit of the learning policy to a given iLQG controller can be computed. Although it cannot be guaranteed that the approximated policy inherits the stability properties of the iLQG controller, the fitting method works well in practice and so far has always led to stable initial learning policies.

Algorithm 1 PI^2-01 Algorithm

Given
- Policy as in (5)
- Cost function $J = h\left(x(t_f)\right) + \sum_{t=0}^{t_f-1} l\left(x(t),u(t)\right)$
- Parameter regularization matrix R

Initialization
- Initialize θ; $\delta\theta \leftarrow \mathbf{0}$,
- $c \leftarrow$ noise initial standard deviation

repeat
- Create K roll-outs of the system with the policy parameter $\theta + \varepsilon, \quad \varepsilon \sim \mathcal{N}(c\delta\theta, c^2\mathbf{I})$
- Reshape $\Psi(t,x)$ to $\begin{bmatrix} g_1 x'^T \\ \vdots \\ g_N x'^T \end{bmatrix}$, $g_i = e^{-\frac{1}{2}\frac{(t-c_i)^2}{\sigma_i^2}}$

for the ith control input **do**
for all the columns of $\Psi(t,x) = [\Psi_j]$ **do**
- Reshape the the ith column of Θ to match $\Psi(t,x)$

for the kth roll-out **do**
for each time, t **do**
- $M^k(t) = \frac{R^{-1}\Psi_j^k \Psi_j^{kT}}{\Psi_j^{kT} R^{-1} \Psi_j^k}$
- $\hat{\theta}(t) = \theta_{ji} + M^k(t)\varepsilon$
- $Q(\tau^k(t)) = h\left(x(t_f)\right) + \sum_t^{t_f-1} l\left(x(t),u(t)\right)$
- $S(\tau^k(t)) = Q(\tau^k(t)) + \sum_t^{t_f-1} \hat{\theta}(t)^T R \hat{\theta}(t)$
- $P(\tau^k(t)) = \frac{e^{-\frac{1}{\lambda}S(\tau^k(t))}}{\sum_{k=1}^K e^{-\frac{1}{\lambda}S(\tau^k(t))}}$

end for
end for
- $w_{ij}(t) = \sum_{k=1}^K P(\tau^k(t))\Psi_j^k$
- $\delta\theta_{ji}(t) = \sum_{k=1}^K P(\tau^k(t)) M^k(t)\varepsilon$

end for
- Reshape $\delta\theta_{ji}(t)$ back to $\delta\theta_i(t)$
- Reshape $w_{ji}(t)$ back to $w_i(t)$
- $\delta\theta_i = \frac{\sum_{t=0}^{t_f-1} w_i(t)\delta\theta_i(t)}{\sum_{t=0}^{t_f-1} w_i(t)}$

end for
- Decrease c for noise annealing

until maximum number of iterations

B. PI^2-01 Algorithm

PI^2-01 has a more general class of policies than PI^2 [5] where DMPs are used for representing the policy. In contrast, the newly introduced PI^2-01 uses a parametrization method similar to the one introduced in [4] for learning impedance control. Instead of using DMPs, the approach assumes a very fast intermediate system. The fast dynamics of this system are practically negligible compared to the system dynamics. Therefore, one can assume that the parametrized policy directly acts on the system inputs. As DMPs are omitted, the update formula of PI^2-01 looks similar to the one in the PoWER algorithm [13]. However, PoWER is more restrictive on the choice of cost functions [5].

Despite the similarity between PI^2-01 and the one in [4], there is a major difference with respect to the controller parametrization and structure. The method presented in [4] uses a conventional PD controller where the reference trajectory and the gain scheduling are learned simultaneously. Also, to keep the number of parameters in the algorithm low, the damping gain is defined as a function of the

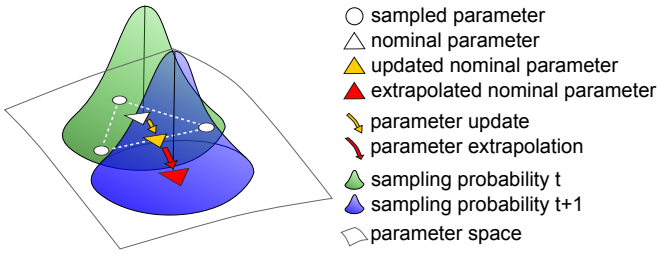

Fig. 1: Illustration of the improved exploration method of PI^2-01. Instead of sampling around the updated parameter, PI^2-01 extrapolates the sampling center in update direction.

proportional gain. Furthermore, the feedforward controller is not included in the learning process. In contrast, here we are learning simultaneously a full state feedback controller plus a feedforward controller. This way of parametrizing the controller blends the reference trajectory design with the feedforward design. As a result, the number of parameters in this algorithm is increased. However, the modification made to the original PI^2 algorithm keeps the implementation still tractable. Algorithm 1 illustrates the pseudo code for the proposed method.

1) Exploration in PI^2-01: An important factor for the success of a learning algorithm is the exploration noise. In the presented algorithm as well as suggested in [6], a constant noise is added to the parameter vector as opposed to a time varying noise. In general, this way of inserting the exploration noise produces smoother trajectories which is an important empirical consideration. However, in contrast to the CMA-ES algorithm which tries to modify the covariance of the noise for more efficient exploration, here the mean of the exploration noise is modified between iterations. This method has an advantage over the modification of the covariance in problems with many parameters and few samples per roll-out. The reason is related to the characteristic of the update rule in the PI^2 algorithm. The updated nominal parameter (white triangle in Figure 1) will always lie within the convex hull (large white dashed triangle) of the sampled parameters (white circles). Therefore, by adding zero mean noise to the updated nominal parameter, new samples will likely lie inside the convex hull as well. Thus, these samples do not provide any new information to the learning algorithm. In contrast to this approach, gradient descent algorithms typically choose new parameters outside the hull of samples which gives these algorithms better exploration capabilities. In the current algorithm, a similar approach is used for exploration. With this improved exploration strategy, the algorithm will benefit from both, the robustness of the PI^2 algorithm and the exploration power of gradient based methods. To implement this idea, the exploration policy distribution is centred at the distance of the noise's standard deviation of the updated parameter (red arrow) in the direction of the latest parameter update (yellow arrow). This approach decreases the probability of choosing samples inside the convex hull.

2) Time-averaging method: In the original PI^2 algorithm the time-averaging weights are defined as a function of kernel activation. Since the function approximation is done on the phase variable rather than time, all the rollouts have the same kernel activation. Contrarily, in PI^2-01 the kernels are trajectory dependent as they are defined over time and state variables. Therefore, the time-averaging weights in PI^2-01 are chosen as the average of the kernel activation over the trajectory probabilities of the corresponding rollout. Furthermore, the time-averaging method of PI^2-01 does not emphasize the parameters corresponding to the beginning of the trajectory but treats them equally throughout the entire trajectory. This approach is better suited for non-DMP based learning. For more details refer to Algorithm 1.

V. RESULTS

In order to compare the performance of existing algorithms as well as the proposed approach several experiments in simulation and on hardware are made. First, we will compare the different approaches in simulation. Then, the proposed approach is used on the ballbot Rezero to prove the practicality of the developed algorithm.

A. Experimental Setup

1) Hardware: The ballbot Rezero used for the experiments consists of three major elements: the ball, the propulsion unit and the upper body. The ball is driven by three omniwheels mounted on brushless motors with planetary gear heads. Through optical encoders on the motors and odometry, the ball rotation is measured. An IMU on the robot estimates the body angles and body angle rates of the robot. Hence, all states are directly measured by on-board sensors.

2) Robot model: For the simulation and the derivation of controllers a non-linear 3D model of the system dynamics of Rezero has been analytically derived [8]. In this paper a slightly simplified model is used in which the wheel dynamics are neglected. This model describes the robot as two rigid bodies: the ball and the upper body. It assumes that no slip and no friction occurs. While the state and the control input are briefly introduced in this section we refer to [8] for the full description of the model.

The state x of Rezero is defined as

$$x = \begin{bmatrix} \vartheta_x & \dot{\vartheta}_x & \vartheta_y & \dot{\vartheta}_y & \vartheta_z & \dot{\vartheta}_z & \varphi_x & \dot{\varphi}_x & \varphi_y & \dot{\varphi}_y \end{bmatrix}^T$$

The state vector x consists of the body angles with respect to gravity (ϑ_x, ϑ_y, ϑ_z) and its derivatives ($\dot{\vartheta}_x$, $\dot{\vartheta}_y$, $\dot{\vartheta}_z$) which represent the angular velocities. Furthermore, the state includes the rotational angles of the ball (φ_x, φ_y) as well as their derivatives ($\dot{\varphi}_x$, $\dot{\varphi}_y$) to represent the ball's position with respect to a reference position (e.g. start position) and the ball's velocity, respectively. $u = [\tau_1, \tau_2, \tau_3]^T$ defines the controller input vector, i.e. the motor torques.

B. Test Cases and Results

1) Cost Function: Throughout all of the experiments the exact same cost function is used. The goal is to control the robot from an initial state to a final state within 6 seconds. At the final time, the center of the ball should be at the $[3, 1]^T$ meter from its starting position and the robot should also have zero velocity, zero tilt angles and tilt angle rates as well

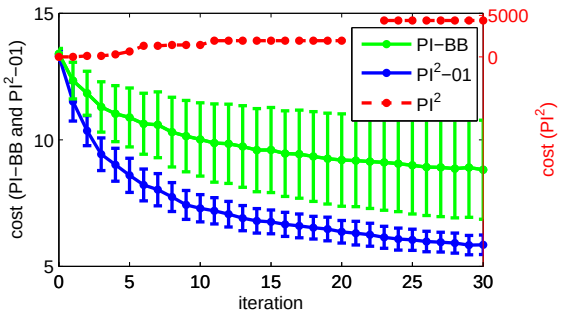

Fig. 2: Comparison of the different learning algorithms in simulation (standard deviation scaled to half). PI^2 (displayed on right y-axis) becomes unstable. PI^2-BB converges but with high variance. PI^2-01 learns faster with low variance and achieves the lowest cost in the test. 10 rollouts (4 new ones, 6 reused) are used per iteration.

as zero change in the heading. The designed cost function has the same form as in (3) where intermediate $l(x,u)$ and final $h(x)$ costs are defined as follows:

$$l(x,u) = 10\vartheta_z^2 + \dot{\vartheta}_z^2 + 1.5u^T u$$
$$h(x) = 10(x - x_d)^T H (x - x_d)$$

where H and x_d are defined as:

$$H = diag(1,2,1,2,1,2,10r^2,4r^2,10r^2,4r^2)$$
$$x_d = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 3/r & 0 & 1/r & 0\end{bmatrix}^T$$

where r denotes the radius of the ball the robot balances on. The numerical parameters of the cost functions are hand-tuned in simulation and validated on the hardware.

2) Policy parametrization: For approximating the time variation of the control inputs, 15 Gaussian functions are used. Therefore, the control policy has in total 495 parameters. Without this re-parametrization the 11 states and 3 control inputs would lead e.g. to 9900 parameters assuming a sampling rate of 50 Hz and a time horizon of 6 seconds.

3) Simulation: In order to validate the proposed algorithm, the performance of the PI^2-01 algorithm is compared with the two state-of-the-art reinforcement algorithms, namely the PI^2 algorithm and PI-BB. To have a fair comparison all the algorithms use iLQG for parameter initialization. Also, to verify that the learning approach is able to adapt the controller to model inaccuracies, different model parameters are used for the design of the iLQG controller and for generating samples. Namely, the total mass and the ball radius of the iLQG model are increased by 10%. One observation during this experiment is the fast convergence of the iLQG method in simulation. The algorithm converges within at most 3 iterations. This feature makes this algorithm a suitable choice for initializing the learning algorithm.

Figure 2 compares the achieved costs of the different learning algorithms. Each iteration consists of 10 rollouts (4 newly generated and 6 best from the previous iteration). The graph shows the mean and standard deviation σ (to enhance the readability we plot 0.5σ) of the learning curves over 10 independent runs of each algorithm.

For PI^2 a constant exploration noise instead of a time-varying noise is used (as suggested in [6]). However, in this experiment, the PI^2 algorithm mostly diverges after few iterations. The main reason for this behavior is the naive time-averaging method in PI^2 which is modified in PI^2-01. But even if the same time-averaging method is used in PI^2, its convergence rate is still lower than PI^2-01, mostly because of the different exploration policy used in PI^2-01.

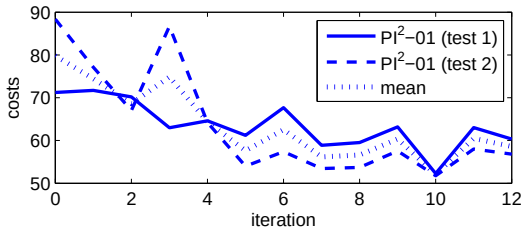

Fig. 3: Two equal learning experiments on hardware using PI^2-01. The plot shows the learning curves for test 1 (solid) and test 2 (dashed). While initial costs vary over the experiments the learned trajectories achieve costs of around 60 (15-30% improvement). Per iteration 8 rollouts (4 new, 4 reused) are used.

The results show that PI^2-01 also outperforms PI-BB with respect to both the convergence rate and the variance of the cost. This comparison emphasizes the importance of using the individual time indexed elements of the cost function rather than just using the total cost accumulated over time.

4) Hardware: To validate the performance of PI^2-01 on real systems, two equal hardware tests are conducted. Both tests are initialized with the same iLGQ controller as optimized in simulation. While this controller, as shown, achieves very low costs in simulation (usually between 10 and 14) the initial performance on the hardware is lower (see iteration 0 in figure 3) and varies due to sensor and actuator noise. The main reason for the cost increase is the difference between the model and the hardware. Many effects like actuator dynamics, gear backlash or the elasticity of the ball are difficult to model and therefore not included. Also, measured or calculated parameters such as inertia deviate from their real values.

After running iLQG, PI^2-01 is initiated. In both tests we are performing 12 iterations of the learning algorithm with 8 rollouts per iteration. From each iteration the 4 best rollouts (out of the total 8) are carried over to the next iteration. The exploration noise is decreased compared to the simulation test as the process noise provides additional exploration.

After each iteration the updated policy is rolled out and the costs of the performance is calculated. Figure 2 shows this cost after each iteration. One observation during the learning is that the sampled rollouts are subject to high process noise. However, the learning algorithm is still able to learn a policy with approximately 15-30% reduced cost within 5 iterations (see Figure 3). In the subsequent iterations rollouts vary less and indicating that the learning has converged.

Figure 4 shows a comparison of the robot's upper body tilt angles with respect to gravity for iLQG and the learned motion in test 2 (results obtained from test 1 are comparable and therefore not shown). PI^2-01 performs the desired motion with smaller tilt angles leading to less overshoot in position (as shown in Figure 5). Furthermore, the learned trajectory

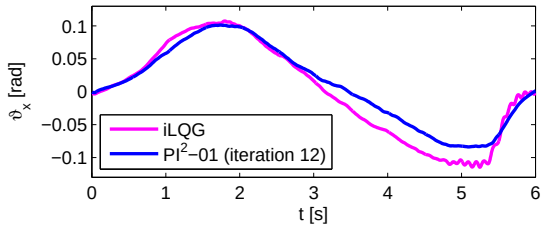

Fig. 4: Rezero's tilt angle ϑ_x for test 1 on hardware. PI^2-01 after 12 iterations produces a smoother trajectory than iLQG.

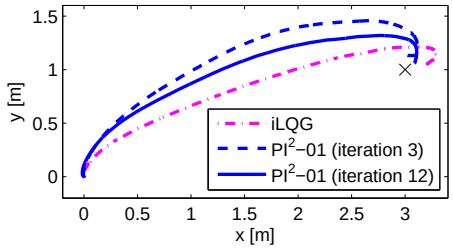

Fig. 5: Trajectory comparison for test 2 on hardware. While iLGQ takes a shorter path than PI^2-01 (iteration 3 and 12), its larger final distance to the goal and the increased overshoot lead to higher costs.

approaches the goal state more gracefully and with lower gains. Hence, chattering due to gear backlash (as seen around $4.8 \leq t \leq 6$ in the motor torques in Figure 6) is not present in the motion control of PI^2-01. Learning tends to avoid chattering as it leads to increased tilt angles and tilt angle rates which are penalized in the cost function. This is a good example for the adaptation of the learning algorithm to an unmodelled (as difficult to model) system property.

VI. CONCLUSIONS

This work presents a general approach for learning motion control. This process consists of two main steps. The first step tries to leverage the information contained in a system model and designs a model-based, iLQG controller. In general the performance of this type of controller depends on the accuracy of the model. As many robots contain non-linear elements (e.g. flexible parts, cable drives or harmonic drives) or are subject to complex non-linear dynamics (e.g. aerodynamics, rigid body dynamics or friction) sufficiently precise system models cannot be provided. Hence, we need to tune the designed controller on the real system. To automate this process, a reinforcement learning approach is proposed. The learning algorithm in this paper is named PI^2-01 which is a variant of the PI^2 algorithm. In order to make the algorithm more robust, the time-averaging of PI^2 is modified. Secondly, through a more sophisticated sampling policy, the exploration of the algorithm is improved.

To verify the performance, the algorithm is implemented on the ballbot Rezero. Tests are performed both in simulation and on the real system. In simulation the PI^2-01 algorithm shows clearly better performance than both the original PI^2 and the PI-BB algorithms. Also, the algorithm is proven to work equally well on the hardware. The fact that the proposed method can learn motion controllers both in simulation and on hardware proves the scalability of the proposed method for systems with medium state size. In the future, we will investigate the scalability of the approach to systems with a higher number of states like legged robots.

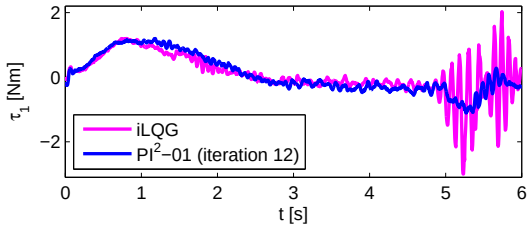

Fig. 6: Torque command of motor 1 for test 1 on hardware. The iLQG controller suffers from chattering due to gear backlash at the end of the trajectory while iteration 12 of PI^2-01 shows a smoother torque output. The other two motors show the same effect.

Finally, this work proves that the combination of a suitable function approximation method and an efficient learning strategy is able to optimize a fairly complex controller in few iterations. The proposed algorithm's ability to learn full state feedback plus feedforward control on a high-dimensional system places it in a group of few but very successful combined motion control approaches.

ACKNOWLEDGEMENT

We gratefully acknowledge Péter Fankhauser and the Autonomous Systems Lab for their support with Rezero.

This research has been funded partially through a Swiss National Science Foundation Professorship award to Jonas Buchli, the NCCR Robotics and the EU Project BALANCE (Grant 601003, EU FP7 program).

REFERENCES

[1] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proc. of the American Control Conference*, 2005.

[2] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, 1966.

[3] F. Stulp, J. Buchli, E. Theodorou, and S. Schaal, "Reinforcement learning of full-body humanoid motor skills," in *Proc. of the IEEE-RAS International Conference on Humanoid Robots*, 2010.

[4] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, "Learning variable impedance control," *The International Journal of Robotics Research*, vol. 30, no. 7, 2011.

[5] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *The Journal of Machine Learning Research*, vol. 11, 2010.

[6] F. Stulp, O. Sigaud, *et al.*, "Policy improvement methods: Between black-box optimization and episodic reinforcement learning," 2012.

[7] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proc. of the 28th International Conference on Machine Learning*, 2011.

[8] P. Fankhauser and C. Gwerder, "Modeling and control of a ballbot," *Bachelor thesis, ETH Zurich*, 2010.

[9] T. Lauwers, G. Kantor, and R. Hollis, "A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2006.

[10] M. Kumagai and T. Ochiai, "Development of a robot balancing on a ball," in *International Conference on Control, Automation and Systems*, 2008.

[11] U. Nagarajan and R. Hollis, "Shape space planner for shape-accelerated balancing mobile robots," *The International Journal of Robotics Research*, vol. 32, no. 11, 2013.

[12] M. Shomin and R. Hollis, "Differentially flat trajectory generation for a dynamically stable mobile robot," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2013.

[13] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Proc. of the Advances in Neural Information Processing Systems*, 2008.