MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Memory-Based Global Iterative Linear Quadratic Control

Nikovski, Daniel; Zhong, Junmin; Yerazunis, William S.

TR2024-089 July 02, 2024

Abstract

We propose a method for designing global nonlinear controllers based on the application of memory-based learning schemes for the purpose of aggregating multiple solutions produced by optimal control algorithms based on differential dynamic programming. The method leverages the fact that these optimal control algorithms produce not only nominal state and control trajectories, but entire full-state feedback (FSF) controllers, and the combined controller effectively switches between these multiple FSF controllers. Empirical verification demonstrates that it can be very effective in solving difficult benchmark control problems at high control rates.

International Conference on Control, Decision and Information Technologies (CoDIT 2024)

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Memory-Based Global Iterative Linear Quadratic Control

Daniel Nikovski, Junmin Zhong and William Yerazunis†

***Abstract*— We propose a method for designing global nonlinear controllers based on the application of memory-based learning schemes for the purpose of aggregating multiple solutions produced by optimal control algorithms based on differential dynamic programming. The method leverages the fact that these optimal control algorithms produce not only nominal state and control trajectories, but entire full-state feedback (FSF) controllers, and the combined controller effectively switches between these multiple FSF controllers. Empirical verification demonstrates that it can be very effective in solving difficult benchmark control problems at high control rates.**

***Index Terms*— Learning control, memory-based learning, optimal control, dynamic programming**

I. Introduction

Computing an optimal feedback controller for an arbitrary nonlinear system is very difficult in the general case, and usually various custom solutions are employed for specific classes of nonlinear systems. In some cases, an optimal trajectory can be computed for a given initial and goal state, and executed in open-loop. This would not work well when disturbances are present, but if the optimal trajectory is re-computed quickly, and only the first control from it applied at each control step, a form of closed loop control can be achieved (commonly called model-predictive control, MPC). However, the success of such MPC schemes often depends on the length of the predictive horizon over which trajectories are computed. For some systems, relatively short horizons are sufficient, but for others, such as non-minimum phase systems where the controller needs to move away from the goal state initially and approach it only later, the recomputation of the entire trajectory to the goal at each control step might be necessary, placing big demands on computing power and effectively reducing the achievable control rate. This limits the applicability of such entirely online methods (also called implicit MPC schemes).

Another, very different approach is to use model-free reinforcement learning algorithms to compute off-line a true universal feed-back control policy by means of repeated trial and error, and execute the policy online. This approach is also known as explicit MPC in the control systems community. A recent incarnation of this idea, known as deep reinforcement learning (DRL) uses deep-neural nets to store the control policy (and possibly, the value function) in a deep neural net, significantly increasing the dimensionality and complexity of problems that can be solved this way. Major recent algorithmic advances in DRL have largely eliminated one of the traditional weaknesses of RL – the inability to deal with continuous state and control spaces, and recent algorithms such as DDPG, TRPO, SAC, A3C, etc. are fully capable of finding good control policies in continuous state and control spaces [1]. The computed policies that are stored in deep neural nets can typically be executed very quickly, allowing for very high control rates. However, this approach has a number of significant shortcomings, too. First, when model-free algorithms are used, the resulting derivative-free optimization methods for finding the optimal policy are excruciatingly slow and data inefficient. Furthermore, computing optimal decisions for all parts of the state space might not even be necessary, depending on how the controller will be used.

A very useful middle ground is occupied by algorithms that make full use of model derivatives to compute optimal state and control trajectories, but in addition also compute locally optimal control laws that are valid in the vicinity of the optimal trajectory, so they are, in fact, closed-loop controllers. Examples of such algorithms are the Differential Dynamic Programming (DDP) algorithm ([2]) and its more modern and computationally efficient variant the Iterative Linear Quadratic Regulator (iLQR) algorithm ([3]). However, their control laws are valid only in a relatively small part of the state space and risk losing control. To combat this, DDP and iLQR can be executed in implicit MPC style, continuously recomputing the trajectory to the goal state, which is still very computationally heavy [4].

To address the shortcomings of both general DRL algorithms and those based on DDP, the Guided Policy Search (GPS) algorithm ([5]) uses a combination of the two that leverages both the fast computation of DDP/iLQR as well as the expressive power and generalization abilities of deep neural networks. The GPS algorithm does this by solving an iLQR problem from multiple starting points, generating a number of training examples matching state to control from these solutions, and loading these examples into a deep neural net by training the net to match the states (or, higher-dimensional observations, if states are not directly observable at run-time), to controls. By leveraging the generalization (essentially, interpolation) abilities of neural nets, a full control policy can be computed over the entire part of the state space covered by the examples. However, using neural nets has the associated problems of long training times and lack of guarantees about convergence. Furthermore, neural nets do not enforce consistency between examples.

The current paper proposes an alternative method for combining multiple local iLQR solutions into one global policy. The method is described in Section II, and its empir-

†All authors are with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, USA 02139 {nikovski,jzhong,yerazunis}@merl.com

ical verification on a test problem is described in Section III. Section IV discusses ideas for extending the method further and concludes the paper. The main benefit of the proposed method is that a full global control policy can be constructed from relatively few solutions from specific starting states, requiring much less computation than typical DRL algorithms.

II. Memory-Based Learning for Constructing Global Nonlinear Control Policies

A. Problem Statement

We consider the problem of stabilizing a fully observable nonlinear time-invariant dynamical system described by the discrete state dynamics equation $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$, where $\mathbf{x}$ is a multidimensional continuous state space, $\mathbf{u}$ is a continuous control vector, and $\mathbf{f}$ is a nonlinear time-invariant vector field. We are primarily concerned with control problems where the objective is to bring the system from an initial state $\mathbf{x}_0$ to a goal state $\mathbf{x}^{(G)}$ in some optimal way. This formulation of the control problem corresponds to both stabilization problems, where $\mathbf{x}^{(G)}$ is a set-point, as well as planning problems, where $\mathbf{x}^{(G)}$ is possibly quite far from the initial state $\mathbf{x}_0$, and reaching the goal cannot always be computed by gradually reducing the feedback error $\mathbf{x}^{(G)} - \mathbf{x}_k$, but requires traversing a complicated trajectory that might temporarily increase the feedback error before bringing it to zero. Instances of such planning problems arise when the system has unstable open-loop dynamics, for example is non-minimum phase, or is underactuated due to control limits or reduced number of control inputs. Computing control laws for such systems has long been studied both in the fields of control systems engineering as well as artificial intelligence and robotics.

The desired optimality of the computed control law is expressed by means of a cumulative cost J_0 that is the sum of running (stage) costs l_r and a final cost l_f, where the summation is computed over a sequence of control steps:

$$J_0(\mathbf{x}_0, U) = \sum_{k=0}^{H-1} l_r(\mathbf{x}_k, \mathbf{u}_k) + l_f(\mathbf{x}_H), \qquad (1)$$

where the states x_k, $k > 0$ follow the dynamics defined above starting from $\mathbf{x}_0$, and $U = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_H\}$ is the control sequence applied over a finite horizon of length H time steps. Here, a finite horizon is needed to avoid infinite cumulative costs. (In contrast, DRL algorithms typically use infinite discounted cumulative costs/rewards.) By providing suitable positive running costs l_r, desired minimum-time objectives can be achieved. The problem of trajectory optimization is usually meant to consist of finding an optimal sequence of controls $U^* = \arg\min_U J_0(\mathbf{x}_0, U)$ from a specific starting state $\mathbf{x}_0$, and not from every state within the state space of the system.

B. Trajectory Optimization Based on Differential Dynamic Programming

The DDP and iLQR algorithms solve this trajectory optimization problem very efficiently when the dynamics $\mathbf{f}$ and stage costs l_r are differentiable ([3], [6]). Starting from an initial guess for the optimal control trajectory, they compute the resulting state trajectory by rolling out the dynamics forward, and then employing Bellman's principle of optimality to compute the optimal controls and partial costs-to-go starting from the goal state and proceeding backwards in time. (This use of back-to-front dynamic programming is the key to the computational efficiency of the procedure, and contrasts with the asynchronous and directionless way Bellman back-ups of the value function are computed in most DRL algorithms.) Once a new improved control sequence is computed, the forward and backward passes are iterated until convergence. This convergence is typically fast, but necessarily only to a local minimum of the cumulative cost. This, in its turn, contrasts with the convergence properties of algorithms such as value and policy iteration, which are guaranteed to converge to a global optimum, at least when the value function and policy are represented in a tabular format [7]. (Although, when deep neural nets are used to represent them, as is the case with modern DRL algorithms, such global convergence can hardly be guaranteed, either.)

As noted above, even though DDP and iLQR computation is fast, it is usually not fast enough for real-time control, if the entire trajectory has to be recomputed at every control step. The highly influential GPS algorithm ([5]) deals with this problem by using iLQR to precompute a large number of trajectories, starting from many initial states, and then using supervised machine learning to learn the mapping $\mathbf{u} = \mu(\mathbf{x})$ from states $\mathbf{x}$ to controls $\mathbf{u}$ that effectively constitutes a complete policy, that is, a global control law.

This approach combines the remarkable approximation and generalization properties of deep neural nets with the high-speed of trajectory optimization based on differential dynamic programming. However, such approximation power does not come without perils. Supervised machine learning algorithms typically minimize the mean squared error (MSE) over the training set and have the unfortunate property of averaging the outputs of two training examples that happen to have the same input. That is, if the training algorithm sees two pairs of states and controls $(\mathbf{x}_i, \mathbf{u}_i)$ and $(\mathbf{x}_j, \mathbf{u}_j)$ such that $\mathbf{x}_i = \mathbf{x}_j$, but $\mathbf{u}_i \neq \mathbf{u}_j$, then the best prediction for that state that minimizes the MSE would be $(\mathbf{u}_i + \mathbf{u}_j)/2$. It might well be the case, though, that the two examples come from different trajectories, and even though both $\mathbf{u}_i$ and $\mathbf{u}_j$ can be suitable controls for this state, their average might not be. For example, one control might prescribe going to the left of an obstacle, the other might prescribe going to the right of it, but their average would mean colliding with the obstacle, and is thus not a good solution.

C. A Memory Based Method for Combining Multiple iLQR Solutions

The proposed method follows the same general idea as that of the GPS algorithm: use multiple iLQR solutions from a representative number of starting states, and fuse them into a global policy by means of a machine learning approximator. Where our method differs from GPS is in which machine

learning method is employed, as well as what components of the iLQR solutions are used.

Instead of using deep neural networks for combining the multiple iLQR solutions, we propose to use a class of memory-based learning (MBL) algorithms that are also often referred to as non-parametric methods in the field of statistics. These methods include the k-nearest neighbor (k-NN), locally weighted learning (LWL), and locally weighted regression (LWR) algorithms that have already found success in the field of learning control ([8]). They also have the distinct advantage that no computational effort needs to be spent on training – rather, all the training data is simply stored in memory, and a local predictive model is quickly constructed for a specific query point (model input) only after this query point has been identified.

The training data set D is organized as a large collection of input-output pairs $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ obtained from all time steps of all iLQR solutions. (If I iLQR solutions have been computed, each of length H time steps, then the data set will contain $N = IH$ examples.) The inputs are states $\mathbf{x}$ obtained from the states of all iLQR solutions, and the outputs $\mathbf{y}$ contain other elements of those solutions. One possibility is that $\mathbf{y} = \mathbf{u}$, i.e. the output of the MBL model is directly the control to be applied when the system is in state $\mathbf{x}$. This arrangement of the training data is often called *direct inverse control* in the field of learning control [8].

Given a training data set D stored in memory and a new query state $\mathbf{x}$, we make a prediction $\hat{\mathbf{y}}(\mathbf{x}) = g(\mathbf{x})$ by constructing a local model g specifically for the new query point $\mathbf{x}$. Most MBL algorithms start with computing the Euclidean distance $d_i = ||\mathbf{x} - \mathbf{x}_i||_2$ between the new query state $\mathbf{x}$ and the inputs $\mathbf{x}_i$, $1 \leq i \leq N$ of all the examples in the database. (The Euclidean distance can be weighted according to the scales of the individual components of the state space, if the scales differ.) Different MBL algorithms use this distance information differently, for example:

- The k-NN algorithm's prediction is the average of the outputs of the k closest points: $\hat{\mathbf{y}}(\mathbf{x}) = g_{kNN}(\mathbf{x}) = \sum_{i=1}^{k} \mathbf{y}^{(i)}/k$, where $y^{(i)}$ is the output of the i-th closest example.
- In locally-weighted learning (LWL), $\hat{\mathbf{y}}(\mathbf{x}) = g_{LWL}(\mathbf{x}) = \frac{\sum_{i=1}^{N} \mathbf{y}_i C(d_i)}{\sum_{i=1}^{N} C(d_i)}$, where $C(d)$ is a suitably chosen kernel, typically rapidly decreasing as the distance d increases [8].
- In locally-weighted learning (LWR), a local regression model of desired order (e.g., linear, quadratic, etc.) is fitted around the query point by weighting the prediction error on all examples according to the computed distances d_i and using a suitable estimation algorithm, such as weighted least squares [8].

Various MBL algorithms provide various trade-offs between prediction accuracy and computation time. Of particular interest as regards our application of MBL to global control policy construction are k-NN methods, due to their fast computation. Data structures such as k-d trees can be used for fast retrieval of the closest neighbors to a query point, and are particularly effective in low- to medium-dimensional query spaces, as their retrieval time scales logarithmically in the number of examples N [9].

Another favorable property of specifically the 1-NN version is that by finding the closest state in any iLQR solution, it will always execute the action for that solution, thus avoiding the averaging problem associated with many other ML algorithms, as discussed above.

If we use as outputs $\mathbf{y}$ only the controls $\mathbf{u}$ from the iLQR solutions, we call this method MBiLQR-A, for Memory-Based iLQR with nearest Action. This method, just like GPS, would have to rely on the approximation abilities of the chosen ML scheme to smoothly approximate between neighboring solutions. An alternative is to make use of the fact that the DDP and iLQR algorithms compute actual feedback controllers of the form $\hat{\mathbf{u}} = \mathbf{u}_k + K_k(\mathbf{x} - \mathbf{x}_k)$, where K_k are feedback gains specifically appropriate for time step k of the particular iLQR solution. When this controller is executed, if the state $\mathbf{x}$ follows the nominal trajectory $\mathbf{x}_k$, the computed control $\hat{\mathbf{u}}$ will also follow exactly the nominal control trajectory $\hat{\mathbf{u}}_k$. However, when $\mathbf{x}$ deviates from the trajectory $\mathbf{x}_k$, for example due to disturbance or real-world dynamics that differ from the model dynamics $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ used by the algorithm, the controller will act to bring the system's state to the nominal trajectory through the gains K_k. This controller is valid typically only in the local neighborhood of the state space trajectory, but this matches very well the principle of operation of MBL schemes: they build a predictive model only in the local neighborhood of a query point.

Based on this reasoning, we propose a second variation of the control construction method, where the outputs $\mathbf{y}$ of the data set D consist of not only the control $\mathbf{u}$ associated with a particular state $\mathbf{x}$ in an iLQR solution, but also the control gains K associated with that state: $\mathbf{y} = [\mathbf{u}, K]$. As MBL methods predict each of their outputs independently, and most of their computational effort is in computing the distances to the training examples, the addition of the gains K among the model's outputs does not change the computational complexity of the method. We call this version of the controller MBiLQR-C, for nearest Controller.

However, when combining several trajectories generated by the iLQR method, the MBiLQR-C controller may encounter difficulties at specific junctions due to abrupt changes and lack of smoothness in the transitions between trajectories. For instance, within a small area around certain points, the control inputs might be contradictory. One remedy to this issue is to ensure that each reference point is chosen only a single time during a single control run. Another approach, described below, is to remove the explicit dependency of the iLQR solution on time, so that nearest-neighbor searches in space will choose between comparable solution elements.

D. Time-Independent Solution of MBiLQR

For finite horizon problems, the optimal control of the LQG problem is linear in the state via a gain matrix, but this gain matrix is time-dependent [10]. This gain matrix can be

computed from the value function, which is time-dependent, too. Consider the linear time-invariant (LTI) discrete dynamic system of the form:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \tag{2}$$

with stage cost $\frac{1}{2}\mathbf{u}_k^T R\mathbf{u}_k + \frac{1}{2}\mathbf{x}_k^T Q\mathbf{x}_k$, $0 \leq k < H$ and final cost $\frac{1}{2}\mathbf{x}_H^T Q_H \mathbf{x}_H$. The value function for each state is a solution to the Riccati equation, which is solved iteratively, back to front. For finite horizon problems, the solution of the value function V_k is

$$V_k = Q + A^T V_{k+1} A - A^T V_{k+1} B(R + B^T V_{k+1} B)^{-1} B^T V_{k+1} A \tag{3}$$

The result is a sequence of quadratic forms for the value function, each valid everywhere in state space (for LTI systems), but different across time. The optimal control law is written as

$$\mathbf{u}_k = -K_k \mathbf{x}_k \tag{4}$$

where $K_k = (R + B^T V_{k+1} B)^{-1} B^T V_{k+1} A$. Through the time-varying value function V_k, the gain matrix K_k and the reference trajectory $\mathbf{u}_k$ are also time-varying, producing different controls for the same state $\mathbf{x}$ at different time steps k. (As is well known, for infinite-horizon problems on LTI systems, the value function is constant for all time steps, that is $V_k = V_{k+1}$ for all $k \geq 0$, and this property is used to find it and the associated feedback gains by solving Equation 3 at a fixed point; this is the foundation of the fundamental LQR method [10].)

The iLQR algorithm operates similarly to the finite-horizon version of the LQR algorithm, iteratively computing the value function back from the terminal state, but using different local dynamics for each time step resulting from local linearization around the state for that time step. It is applicable only to the finite-horizon setting, because it solves the problem numerically over a fixed horizon. However, even if it was possible to solve it in the infinite horizon setting, there will be different value functions around every state, because the dynamics change over the state space. Furthermore, the optimal control for the same state at different times do not have to be consistent, because the policy is time dependent. Consider the feedback controller of MBiLQR $\hat{\mathbf{u}} = \mathbf{u}_k + K_k(\mathbf{x} - \mathbf{x}_k)$. For the same system state $\mathbf{x}$, differences in the reference control trajectory $\mathbf{u}_k$, reference state trajectory $\mathbf{x}_k$, and gain matrix K_k may arise across different solution trajectories. This variation implies that MBiLQR's output $\hat{\mathbf{u}}(\mathbf{x})$ at state $\mathbf{x}$ could vary abruptly when switching from one controller to another, introducing potential challenges regarding the system's stability, robustness, and convergence. These inconsistencies, stemming from the solution's time-varying nature across different trajectories, could significantly impact MBiLQR's performance and reliability.

One possible solution involves integrating the iLQG algorithm's finite-horizon path from the initial to the goal state with the classical LQR method for infinite-horizon objectives at the goal state. The solution consists of linearizing the dynamics around the goal state and assuming that the terminal value function V_H of the iLQR solver is the solution of the infinite horizon LQR regulation problem for the LTI system with these linearized dynamics. Essentially, the proposed solution constructs an infinite-horizon optimal control problem consisting of two parts: the first part starts at the initial state and spans the first H steps, and the second part consists of stabilizing the system linearized around the goal state from time step $H+1$ to infinity. By doing so, we are modifying the iLQR algorithm to essentially compute a time-invariant solution of the optimal control problem; if it operates on an LTI system to begin with, it will simply compute (iteratively) the constant value function and gain matrix that the LQR method computes analytically. This strategy is predicated on the assumption that the goal will be reached within a period no longer than H steps, after which it will be perpetually maintained. In practice, to convert time-dependent gains to time-invariant ones, the following steps are taken:

- Execute the original iLQR algorithm to compute a trajectory $\mathbf{x}_k$, $0 \leq k \leq H$ such that the final state $\mathbf{x}_H$ necessarily reaches (the neighborhood of) the goal state $\mathbf{x}^{(G)}$.
- Linearize the dynamics around the goal state $\mathbf{x}^{(G)}$, producing matrices A_G and B_G, and cost functions Q_G and R_G.
- Compute analytically the steady-state value-function V_G of the corresponding LQR problem with matrices A_G and B_G and cost functions Q_G and R_G, for example by solving Equation 3 for $V_G = V_k = V_{k+1}$, plugging it on both sides of the equation.
- Conduct an additional backward recursion step of iLQR by setting the iLQR final cost $V_H = V_G$.

III. Empirical Evaluation

A. Torque-Limited Pendulum (TLP) benchmark

In this section, we evaluate empirically the performance of some of the variants of the proposed method on a classical benchmark problem from the control systems literature: the task of swinging up and stabilizing a torque-limited pendulum (TLP) to and around its upper unstable equilibrium. The low dimensional state space of the task is suitable for visualizing the computed control policies, and the need for reaching and stabilizing around an unstable equilibrium makes the task quite difficult for traditional control methods.

The pendulum, shown in Fig. 1, is governed by the equation $mL\ddot{\theta} = -mg\sin\theta - b\dot{\theta} + \tau$, where θ is its angle with respect to the stable vertical hanging position, m is the mass of its bob, L is its length, g is Earth's gravity, b is a viscous friction coefficient, and τ is the applied torque about the point the pendulum is suspended from. The goal is to swing it up from hanging position in the neighborhood of its lower stable equilibrium $\theta = 0$ to its upper unstable equilibrium $\theta = \pi$ and balance it there. Given enough torque, this is not difficult, as the torque can be applied against gravity. Once the pendulum reaches the unstable upper equilibrium, it can be stabilized there by a linear feedback controller, such as a

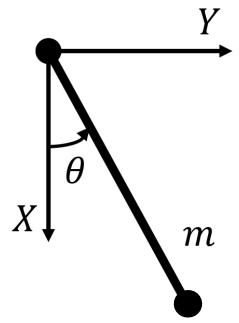

Fig. 1. Torque Limited Pendulum

Params	Length L	Mass m	Damping b	Torque Limit τ_{max}
Value	0.61 m	0.15 kg	0.05 Ns/m	[-0.4,0.4] Nm

TABLE I
PARAMETERS OF THE TLP

PID controller or an LQR controller based on a linearization of the dynamics around the upper equilibrium.

However, when the torque is limited, the controller must pump enough energy into the pendulum by swinging it back and forth one or more times. This essentially turns the problem into a planning one. The analytical construction of such a controller/planner is not trivial, and insights into the physics of the system are necessary for a successful solution [11]. This makes it a suitable benchmark for general-purpose controller-design methods such as the one proposed in this paper.

We verified the algorithm on a TLP simulated in the MuJoCo physics engine [12]. The detailed simulation parameters of the TLP are shown in Table I.

B. Computation of Nominal iLQR Solutions

The first step in all variations of the proposed algorithm is to compute a set of I nominal iLQR solutions starting from multiple starting states. The success of the global control method we propose (and also of the GPS algorithm, for that matter) critically depends on the ability to find these local solutions reliably. The starting states were sampled from the subset of the state space such that $-\pi/2 \leq \theta \leq \pi/2$ (rad) and $-3 \leq \dot{\theta} \leq 3$ (rad/s), reflecting that the objective of the task is to swing up the pendulum from a position generally below its suspension point.

Following the popular practice in the field of learning control when learning models of systems with rotational degrees of freedom expressed by angles, we represent the pendulum's angle θ with its sine and cosine. This avoids angle wrap-around at $\theta = \pm\pi$ and ensures continuity of functions on the angle there, which is assumed by most ML methods. As a result, the state space used by the iLQR algorithm is three dimensional: $\mathbf{x} = [\sin\theta, \cos\theta, \dot{\theta}]$. The control space is one-dimensional: $\mathbf{u} = \tau$.

Each execution of the iLQR algorithm consisted of 100 iterations of the algorithm, initialized with a completely random guess for the nominal control trajectory $\mathbf{u}[k]$, $0 \leq k \leq H-1$, where $H = 200$ time steps. We used quadratic running and terminal costs: $l(\mathbf{x}, \mathbf{u}) = (\mathbf{x} - \mathbf{x}_g)^T Q(\mathbf{x} - \mathbf{x}^{(G)}) + \mathbf{u}^T R\mathbf{u}$ and $l_f(\mathbf{x}_H) = (\mathbf{x}_H - \mathbf{x}^{(G)})^T Q_f(\mathbf{x}_H - \mathbf{x}^{(G)})$, where the goal state, corresponding to the upper unstable equilibrium is $\mathbf{x}^{(G)} = [\sin(\pi), \cos(\pi), 0] = [0, -1, 0]$. We found that the following matrices produced reliable iLQR solutions that found a way to reach the goal state: $Q = diag([10, 100, 1]$, $R = [0.01]$, $Q_f = diag([10, 1000, 1000]$. The very low control cost signifies that the controller is free to saturate the control input while swinging up the pendulum (resulting in bang-bang control), which is known to be optimal in minimum-time problems with control limits. (Our implementation of the iLQR algorithm is aware of control limits and takes them into consideration when computing nominal solutions; it is based on [3].)

A single execution of the iLQR algorithm in this setting took on average of 7.94 s on an i7-10750H CPU, implemented in Python. This number suggests that recomputing the entire trajectory at every control step would be way too slow for real-time control even if the implementation is optimized, but is otherwise acceptable for off-line construction of a training database.

Once an iLQR solution has been computed, we decide whether it has reached the goal state by computing the distance $d = ||\mathbf{x}_H - \mathbf{x}^{(G)}||_2$ between its terminal state $\mathbf{x}_H$ and the desired goal state $\mathbf{x}^{(G)}$, and add the trajectories to the dataset only if this distance is below a threshold ϵ: $d \leq \epsilon$. We used a threshold of $\epsilon = 0.2$. We should point out that, in general, a full-state feedback (FSF) controller without integral action cannot always eliminate steady-state error, so many iLQR solutions converge to a terminal state with some steady-state error, where the pendulum is propped by a small amount of torque at an angle very close to the upper unstable equilibrium. We consider such solutions successful, as the FSF controller with terminal gains K_H will be able to reject disturbances around the upper equilibrium, and keep the pendulum in the goal region.

C. Empirical Results

Consequently, we adopted the same criterion for success when the MBiLQR algorithm is run from a new starting point – whether the controller could bring the pendulum to the goal region, such that $||\mathbf{x}_H - \mathbf{x}^{(G)}||_2 \leq \epsilon$. We conducted $1,000$ test executions from the same subset of the state space used for training, and the execution of one of them is superimposed on the training iLQR solutions in Fig. 2. It is visible that the iLQR solutions do not necessarily cover the state space uniformly, but tend to define a general preferred solution, even though they were computed completely independently from one another, with completely random initialization.

We also evaluated the fraction of successes across the $1,000$ test runs across 5 different random seeds, for a total of $5,000$ random test cases) as a function of how many iLQR solutions the MBiLQR algorithm was working with. The results, shown in Figs. 3, demonstrate that the basic

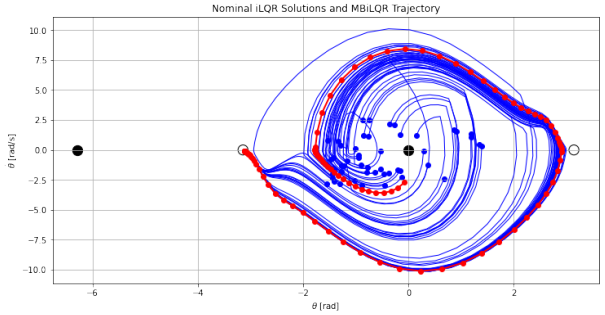

Fig. 2. An example execution trajectory of MBiLQR solution (in red) superimposed on the training iLQR solutions it was was based on (in blue). The stable equilibria are shown with solid dots, and the unstable equilibria (the task's goal state) are shown with hollow dots.

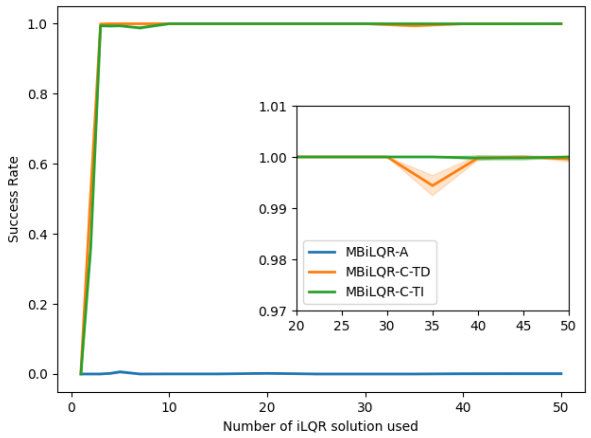

Fig. 3. Success rate vs. the number of iLQR solutions used by MBiLQR-A, MBiLQR-C-TD, MBiLQR-C-TI. The shaded area represents the standard deviation of the success rate across 5 different random seeds.

version of the algorithms, MBiLQR-A, has very minimal success rate, reaching the goal only occasionally. In contrast, the more advanced MBiLQR-C time dependent (MBiLQR-C-TD) version of the algorithm shows the ability to bring the system to the goal state with very high success rates with even very few nominal iLQR solutions. However, it is not necessarily the case that more solutions monotonically increase the success rate. Furthermore, it shows some variance across different random seeds. In contrast, the MBiLQR time independent (MBiLQR-C-TI) version of the algorithm not only shows very high success rate, but has almost no variance across different random seeds. This demonstrates the clear advantage of using time-independent gains.

Note that each experiment with more iLQR solutions in memory included the exact same solutions as those experiments with fewer solutions before it, so any change in performance is due entirely to the additional solutions it is using. Furthermore, the control step computation time never exceeded 0.2 ms, even for the maximum number of solutions, allowing control rates in excess of 5 kHz.

IV. Conclusion and Future Work

We presented a method for designing global nonlinear controllers based on the application of memory-based learning schemes for the purpose of aggregating multiple solutions produced by optimal control algorithms based on differential dynamic programming. The method leverages the fact that these optimal control algorithms produce not only nominal state and control trajectories, but entire FSF controllers, and the combined controller effectively switches between these multiple FSF controllers. Another way to look at the proposed controller is as a method for automatic gain scheduling based on multiple local solutions. Empirical verification demonstrated that it can be very effective in solving difficult benchmark control problems, whereas naive application of MBL that uses only the computed control trajectories and discards the FSF gains was not successful.

Although efficient data structures can be used to shorten dramatically the nearest-neighbor search, it is also true that an excessive number of iLQR solutions in memory will eventually slow down control computation. As many of the solutions might be redundant, as evidenced by their converging trajectories, an appealing possibility for future work is to apply algorithms for exemplar learning, where multiple memory instances are replaced with a single instance that represents all of them.

References

[1] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 12 348–12 355, 2021.

[2] D. H. Jacobson and D. Q. Mayne, *Differential dynamic programming*. Elsevier, 1970.

[3] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *First International Conference on Informatics in Control, Automation and Robotics*, vol. 2. SciTePress, 2004, pp. 222–229.

[4] N. Bruchon, G. Fenu, G. Gaio, S. Hirlander, M. Lonza, F. A. Pellegrino, and E. Salvato, "An Online Iterative Linear Quadratic Approach for a Satisfactory Working Point Attainment at FERMI," *Information*, vol. 12, no. 7, p. 262, 2021.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, pp. 1–40, 2016.

[6] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4906–4913.

[7] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[8] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," *Artificial Intelligence Review*, vol. 11, pp. 75–113, 1997.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, Michel V., B. Thirion, O. Grisel, M. Blondel, Prettenhofer P., R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] R. F. Stengel, *Optimal control and estimation*. Courier Corporation, 1994.

[11] K. J. Åström and K. Furuta, "Swinging up a pendulum by energy control," *Automatica*, vol. 36, no. 2, pp. 287–295, 2000.

[12] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.