Reactive Anticipatory Robot Skills with Memory

Hakan Girgin, Julius Jankowski, and Sylvain Calinon

Idiap Research Institute, Martigny 1920, Switzerland,
{hakan.girgin, julius.jankowski, sylvain.calinon}@idiap.ch

Abstract. Optimal control in robotics has been increasingly popular in recent years and has been applied in many applications involving complex dynamical systems. Closed-loop optimal control strategies include model predictive control (MPC) and time-varying linear controllers optimized through iLQR. However, such feedback controllers rely on the information of the current state, limiting the range of robotic applications where the robot needs to remember what it has done before to act and plan accordingly. The recently proposed system level synthesis (SLS) framework circumvents this limitation via a richer controller structure with memory. In this work, we propose to optimally design reactive anticipatory robot skills with memory by extending SLS to tracking problems involving nonlinear systems and nonquadratic cost functions. We showcase our method with two scenarios exploiting task precisions and object affordances in pick-and-place tasks in a simulated and a real environment with a 7-axis Franka Emika robot.

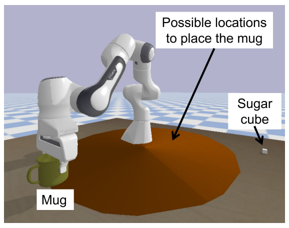
Keywords: optimal control, feedback control with memory, robot control

1 Introduction

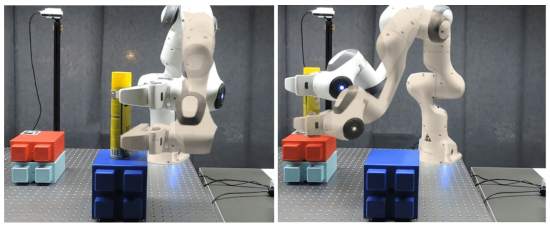
Optimal control can be found in a great variety of applications from economics [1] to engineering problems such as energy management [2] and robotics [3, 4]. It consists of determining optimal actions in problems following a known forward model that describe state changes in time when actions are applied.

In robotics, optimal control has been applied in many applications such as biped walking generation [5, 6], centroidal dynamics trajectory [7, 8] and whole-body motion planning [9]. These works often consider the solution of optimal trajectories of states and actions as a plan over a horizon, which is then tracked with lower level feedback control mechanisms. As the forward models that define these actions are not perfect representations of the real physical movements, feedback control is designed to achieve the planned motion even in the presence of noise, delays and unpredictable perturbations in the environment.

Among others, model predictive control (MPC) [10] is a powerful feedback mechanism in optimal control, that became a key methodology to control complex dynamical systems such as humanoids [11]. It allows to compute the optimal plan over a short receding horizon, apply the first few control commands until a new plan is recomputed using the current state of the robot. However, due to



(a) Mug-sugar cube scenario



(b) Pick-and-place task

Fig. 1: (a) The robot decides where to place the object according to the initial position and with the anticipation to pick up the sugar cube and put it inside the cup. (b) Pick-and-place task with the Franka Emika robot deciding autonomously to grasp the object from different parts, as an adaptation to different initial configurations. The memory feedback property allows the robot to remember where it placed the mug and grasped the object, so that it can place it accordingly.

scaling and performance issues of the solvers, MPC approaches still have many challenges to be solved for high dimensional systems, for longer receding horizons, and for high frequency control. In this work, we tackle these performance issues by exploiting the structure and output of the solvers as much as possible, to replan faster in the presence of physical perturbations and changes in the task.

Linear quadratic tracking (LQT) [12] and iterative linear quadratic regulator (iLQR) [13] and their variants are simple solvers that are increasingly used in MPC frameworks for controlling high dimensional robotic systems such as humanoids and manipulators [14, 15, 11, 16]. In particular, dynamic programming solutions for these methods are often computationally less intensive than other trajectory optimization methods, with the additional benefit of outputting a full state feedback controller with gains over the horizon. The feedback controller structure in iLQR enables the feedback gains to react to the current state of the system while anticipating the future states. The current state contains implicitly the information from past states; however, the controller cannot directly react on this past information explicitly, resulting in mediocre performance in applications where the robot needs to remember what it has done before to react and replan. For example, a robot grasping an object from its different parts needs to remember where it grasped the object to place it without collision as illustrated in Figure 1b. One solution is to augment the state-space with the past states and solve the problem in the standard way. However, it would result in a much higher dimensional problem which may be impractical for robotic applications.

Recently, System Level Synthesis (SLS) [17] was proposed as an optimal controller reacting not only to the current state but also to the past states of the system through closed-loop mappings optimization. Such a controller with memory could be exploited in tasks requiring correlations between states at different timesteps, as opposed to a controller without memory such as LQR controller. However, SLS framework has a couple of limitations to be used in robotics applications requiring a memory of states with sparse cost functions and nonlinear dynamics.

In this work, we investigate how the optimal control framework can produce anticipatory and reactive robot skills with a memory of the states to produce smart behaviors that can exploit task precisions and object affordances. We propose to achieve our goal with the following extensions to the SLS framework: 1) by adding a feedforward part to the controller enabling trajectory and viapoint tracking, 2) by extending the approach to nonquadratic costs and nonlinear systems using a Newton method optimization; and 3) by providing fast adaptation and replanning strategies.

The paper is organized as follows. First, Section 2 presents the related work in controllers with memory and SLS controllers in robotics. In Section 3, we introduce the SLS framework as background. Then, in Section 4, we show our proposed extensions over SLS that enable its application in robotics and the proposed adaptation strategies when the robot encounters new situations. In Section 5, we showcase our approach with two scenarios exploiting task precisions and object affordances in pick-and-place tasks in a simulated and a real environment with a 7-axis Franka Emika robot. Finally, we conclude the work and discuss potential impacts of the work in Section 6.

2 Related Work

Feedback controllers that can act on the history of states are investigated in robotics mainly via reinforcement learning algorithms by changing the structure of the policy, especially with recurrent neural networks (RNN) [18]. In [19], the authors propose to learn stable bipedal locomotion with an RNN policy, where the memory serves as a model of the physical parameters of the task. In [20], a deep learning architecture for learning a policy that can remember some important information from the past observations is developed by augmenting the policy state with a continuous memory state. The authors showed that their guided policy search algorithm could solve a peg sorting task with a manipulator which needs to remember the target hole position given at the beginning of the training and a plate and bottle placing task where the robot needs to remember which object it is holding to determine the orientation.

System level synthesis (SLS) has been developed in a collection of works that are summarized in [17]. It provides a novel perspective on robust optimal control design by optimizing over the closed-loop mappings of the system, instead of directly optimizing the controller. The authors showed how SLS can be exploited in the domain of large-scale distributed optimal control and robust optimal control.

In [21], SLS has been exploited to learn unknown dynamics for safe control with LQR, including robust safety guarantees in the state and input constraints. In [22], perception based SLS controllers are designed for autonomous control of vehicles learning linear time invariant dynamical systems from image data. In [23], robust perception-based SLS controller is applied for the safe control of a quadrotor. The work in [24] gives necessary and sufficient conditions for the existence of SLS controller for nonlinear systems and [25] exploits these ideas by designing a nonlinear controller for a constrained LQR problem by blending several linear controllers.

3 Background

This section follows [17] to present the SLS framework for regulation problems. A linear-time-varying (LTV) system can be written as $\boldsymbol{x}_{t+1} = \boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{w}_t, \forall t{=}\{0,\dots,T\}$, where $\boldsymbol{x}_t \in \mathbb{R}^m$ is the state, $\boldsymbol{u}_t \in \mathbb{R}^n$ is the control input, and $\boldsymbol{w}_t \in \mathbb{R}^m$ is an exogenous disturbance term. By stacking these vectors for each time-step t, we define $\boldsymbol{x}{=}\left[\boldsymbol{x}_0^\top\ \boldsymbol{x}_1^\top\ \dots\ \boldsymbol{x}_T^\top\right]^\top$, $\boldsymbol{u}{=}\left[\boldsymbol{u}_0^\top\ \boldsymbol{u}_1^\top\ \dots\ \boldsymbol{u}_T^\top\right]^\top$ and $\boldsymbol{w}{=}\left[\boldsymbol{x}_0^\top\ \boldsymbol{w}_0^\top\ \boldsymbol{w}_1^\top\ \dots\ \boldsymbol{w}_{T-1}^\top\right]^\top$, which allows us to express the dynamics as

$$\boldsymbol{x} = \boldsymbol{Z}\boldsymbol{A}_d\boldsymbol{x} + \boldsymbol{Z}\boldsymbol{B}_d\boldsymbol{u} + \boldsymbol{w}, \tag{1}$$

where $\boldsymbol{Z}$ is a delaying operator with identity matrices along its first block sub-diagonal and zeros elsewhere, $\boldsymbol{A}_d{=}\,\mathrm{blkdiag}\,(\boldsymbol{A}_0, \boldsymbol{A}_1, \dots, \boldsymbol{A}_T)$ and $\boldsymbol{B}_d{=}\,\mathrm{blkdiag}(\boldsymbol{B}_0, \boldsymbol{B}_1, \dots, \boldsymbol{B}_T)$. In this work, we consider the stacked disturbance as $\boldsymbol{w}{\sim}\mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$, where $\boldsymbol{\mu}_w{=}[\boldsymbol{\mu}_{x_0}^\top, \boldsymbol{0}^\top]^\top$ and $\boldsymbol{\Sigma}_w{=}\mathrm{blkdiag}(\boldsymbol{\Sigma}_{x_0}, \boldsymbol{\Sigma}_{\text{noise}})$.

We assume a controller of the form $\boldsymbol{u}{=}\boldsymbol{K}\boldsymbol{x}$, where $\boldsymbol{K}$ is a lower block triangular matrix. Note that this controller is more advanced than the controller found by linear quadratic regulator (LQR). In fact, the former includes the latter at its block-diagonal elements while the off-block-diagonal elements act on the history of the states. Inserting the controller definition into (1), we get $\boldsymbol{x}{=}\boldsymbol{Z}\boldsymbol{A}_d\boldsymbol{x} + \boldsymbol{Z}\boldsymbol{B}_d\boldsymbol{K}\boldsymbol{x} + \boldsymbol{w}$, which results in the closed-loop responses

$$\boldsymbol{x} = \boldsymbol{\Phi}_x\boldsymbol{w}, \quad \boldsymbol{\Phi}_x = \Big(\boldsymbol{I} - \boldsymbol{Z}(\boldsymbol{A}_d + \boldsymbol{B}_d\boldsymbol{K})\Big)^{-1}, \tag{2}$$

$$\boldsymbol{u} = \boldsymbol{\Phi}_u\boldsymbol{w}, \quad \boldsymbol{\Phi}_u = \boldsymbol{K}\Big(\boldsymbol{I} - \boldsymbol{Z}(\boldsymbol{A}_d + \boldsymbol{B}_d\boldsymbol{K})\Big)^{-1}, \tag{3}$$

where $\{\boldsymbol{\Phi}_x, \boldsymbol{\Phi}_u\}$ describe the closed loop system responses from the exogenous disturbance $\boldsymbol{w}$ to the state $\boldsymbol{x}$ and control input $\boldsymbol{u}$, respectively. SLS optimizes directly over these system responses, instead of the controller map $\boldsymbol{K}$ as was done by the dynamic programming solutions of LQR. As the controller $\boldsymbol{K}$ is a block lower triangular matrix, these maps are also lower block triangular matrices.

By inserting (2) and (3) into the dynamics equation (1), we obtain

$$\begin{aligned} \boldsymbol{\Phi}_x\boldsymbol{w} &= \boldsymbol{Z}\boldsymbol{A}_d\boldsymbol{\Phi}_x\boldsymbol{w} + \boldsymbol{Z}\boldsymbol{B}_d\boldsymbol{\Phi}_u\boldsymbol{w} + \boldsymbol{w}, \\ \implies \boldsymbol{\Phi}_x &= \boldsymbol{Z}\boldsymbol{A}_d\boldsymbol{\Phi}_x + \boldsymbol{Z}\boldsymbol{B}_d\boldsymbol{\Phi}_u + \boldsymbol{I} = \boldsymbol{S}_{\boldsymbol{x}} + \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{\Phi}_u, \end{aligned} \tag{4}$$

where the implication is because we want this pair of system responses to remain independent of the noise in the system for every initial state, and $\boldsymbol{S}_{\boldsymbol{x}}{=}(\boldsymbol{I} - \boldsymbol{Z}\boldsymbol{A}_d)^{-1}$, $\boldsymbol{S}_{\boldsymbol{u}}{=}\boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{Z}\boldsymbol{B}_d$. The objective of SLS is to optimize directly over these lower block triangular system responses $\{\boldsymbol{\Phi}_x, \boldsymbol{\Phi}_u\}$ so that there exists a linear controller $\boldsymbol{K}$ such that $\boldsymbol{K}{=}\boldsymbol{\Phi}_u\boldsymbol{\Phi}_x^{-1}$ and (4) holds as outlined by the Theorem 2.1 of [17].

Linear quadratic regulator: We consider now a linear quadratic regulator problem with random noise in the process and with a random initial state as

$$\begin{array}{c}\min_{\boldsymbol{x}_t,\boldsymbol{u}_t} \sum_{t=0}^{T} \mathbb{E}[\boldsymbol{x}_t^\top \boldsymbol{Q}_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top \boldsymbol{R}_t \boldsymbol{u}_t] \\ \text{s.t. } \boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{w}_t.\end{array} \tag{5}$$

Using the stacked vectors $\boldsymbol{x}$, $\boldsymbol{u}$ and $\boldsymbol{w}$ and the block diagonal matrices $\boldsymbol{Q} = \text{blkdiag}(\boldsymbol{Q}_0, \boldsymbol{Q}_1, \dots, \boldsymbol{Q}_T)$ and $\boldsymbol{R}{=}\text{blkdiag}(\boldsymbol{R}_0, \boldsymbol{R}_1, \dots, \boldsymbol{R}_T)$, this problem can be recast as optimization over system responses as

$$\begin{array}{c}\min_{\boldsymbol{\Phi}_x,\boldsymbol{\Phi}_u} \mathbb{E}[\|\boldsymbol{\Phi}_x \boldsymbol{w}\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{\Phi}_u \boldsymbol{w}\|_{\boldsymbol{R}}^2] \\ \text{s.t. } \boldsymbol{\Phi}_x = \boldsymbol{S}_x + \boldsymbol{S}_u \boldsymbol{\Phi}_u, \\ \boldsymbol{\Phi}_x, \boldsymbol{\Phi}_u \in \mathcal{L}\end{array} \tag{6}$$

where $\mathcal{L}$ represents the space of lower block triangular matrices. Solving (6), we obtain a feedback controller achieving the desired system responses. One can show that by removing the expectation and treating the cost function as random valued, one can solve the problem independently of the value of $\boldsymbol{w}$. Therefore, in the remainder of this paper, we take the cost functions as random valued cost functions instead of their expectations.

Note that because of the constraint to lie on the space $\mathcal{L}$, this problem cannot be solved directly as we would solve a simple LQR problem. Instead, we make use of the column separability of the objective function and the equality constraint representing the dynamics model to separate the problem into T independent subproblems containing only the nonzero part of each block column of the system responses as in [17].

The controller found by solving (6) can be applied directly to problems with linear forward models and quadratic costs, if the task is to regulate the state to a set-point or to a reference trajectory, which is already dynamically feasible by the plant. In fact, in these cases, one can transform the forward state dynamics to forward error dynamics and still exploit SLS to solve such tasks. However, in most of the robotics applications, the reference trajectory is composed of sparse viapoints and/or the system is nonlinear and/or the cost is nonquadratic. In the next section, we show how to alleviate these problems as part of our contributions.

4 Methods

In this section, we show that we can solve SLS problems for tracking analytically and how to solve them for nonlinear systems and nonquadratic costs. In Section 4.1, we extend the SLS method to *extended system level synthesis* (eSLS) by adding a feedforward part that describes the desired states and the desired control commands. Then, in Section 4.2, we exploit eSLS to solve the problems with nonquadratic cost and nonlinear dynamics, resulting in an *iterative system level synthesis* (iSLS) approach, which greatly extends the domain of applications of the standard SLS.

4.1 Extended system level synthesis (eSLS)

The closed loop responses in (2) and (3) cannot represent the closed-loop dynamics well when the problem is to track a desired state $\boldsymbol{x}_{(d,t)}$ and a desired control command $\boldsymbol{u}_{(d,t)}$ by minimizing $J{=}\mathbb{E}[\|\boldsymbol{x}-\boldsymbol{x}_d\|^2_{\boldsymbol{Q}}+\|\boldsymbol{u}-\boldsymbol{u}_d\|^2_{\boldsymbol{R}}]$ In this section, we propose a new closed-loop response model that can explicitly encode the desired states and the control commands by extending the linear feedback controller in SLS to include also a feedforward term as $\boldsymbol{u}=\boldsymbol{K}\boldsymbol{x}+\boldsymbol{k}$. Inserting this into the dynamics equation (1), we obtain $\boldsymbol{x}=\boldsymbol{Z}\boldsymbol{A}_d\boldsymbol{x}+\boldsymbol{Z}\boldsymbol{B}_d(\boldsymbol{K}\boldsymbol{x}+\boldsymbol{k})+\boldsymbol{w}$, which results in the closed-loop responses

$$\boldsymbol{x}=\boldsymbol{\Phi}_x\boldsymbol{w}+\boldsymbol{d}_{\boldsymbol{x}},\quad \boldsymbol{d}_{\boldsymbol{x}}=\boldsymbol{\Phi}_x\boldsymbol{Z}\boldsymbol{B}_d\boldsymbol{k}, \tag{7}$$

$$\boldsymbol{u}=\boldsymbol{\Phi}_u\boldsymbol{w}+\boldsymbol{d}_{\boldsymbol{u}},\quad \boldsymbol{d}_{\boldsymbol{u}}{=}\boldsymbol{K}\boldsymbol{d}_{\boldsymbol{x}}+\boldsymbol{k}, \tag{8}$$

where $\boldsymbol{\Phi}_x$ and $\boldsymbol{\Phi}_u$ are defined as in (2) and (3). When we insert $\boldsymbol{x}$ and $\boldsymbol{u}$ in (7) and (8) into the dynamics equation (1), we get the dynamics constraints on our optimization variables from $\boldsymbol{\Phi}_x\boldsymbol{w}+\boldsymbol{d}_{\boldsymbol{x}}{=}\boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{w}+\boldsymbol{S}_{\boldsymbol{u}}(\boldsymbol{\Phi}_u\boldsymbol{w}+\boldsymbol{d}_{\boldsymbol{u}})$, which is satisfied for any noise in the system by (4) and $\boldsymbol{d}_{\boldsymbol{x}}=\boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{d}_{\boldsymbol{u}}$. This can be used to show that $\boldsymbol{k}{=}(\boldsymbol{I}-\boldsymbol{K}\boldsymbol{S}_{\boldsymbol{u}})\boldsymbol{d}_{\boldsymbol{u}}$. By a few manipulations of equations, one can rewrite the controller into the more interpretable form of $\boldsymbol{u}=\boldsymbol{K}\boldsymbol{x}+\boldsymbol{k}=\boldsymbol{K}(\boldsymbol{x}-\boldsymbol{d}_{\boldsymbol{x}})+\boldsymbol{d}_{\boldsymbol{u}}$, where $\boldsymbol{d}_{\boldsymbol{x}}$ and $\boldsymbol{d}_{\boldsymbol{u}}$ can be interpreted as the planned trajectory of states and control commands, respectively, assuming zero noise and zero initial state. The feedback part drives the system to a new plan when there is noise, perturbations or nonzero initial state.

The final convex optimization problem becomes

$$\begin{aligned}\min\nolimits_{\boldsymbol{\Phi}_x,\boldsymbol{\Phi}_u,\boldsymbol{d}_{\boldsymbol{x}},\boldsymbol{d}_{\boldsymbol{u}}}\ & \|\boldsymbol{\Phi}_x\boldsymbol{w}+\boldsymbol{d}_{\boldsymbol{x}}-\boldsymbol{x}_d\|^2_{\boldsymbol{Q}}+\|\boldsymbol{\Phi}_u\boldsymbol{w}+\boldsymbol{d}_{\boldsymbol{u}}-\boldsymbol{u}_d\|^2_{\boldsymbol{R}}\\ \text{s.t. } & \boldsymbol{\Phi}_x=\boldsymbol{S}_x+\boldsymbol{S}_u\boldsymbol{\Phi}_u,\\ & \boldsymbol{d}_{\boldsymbol{x}}=\boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{d}_{\boldsymbol{u}},\\ & \boldsymbol{\Phi}_x,\boldsymbol{\Phi}_u\in\mathcal{L}\end{aligned} \tag{9}$$

which can be solved analytically for the optimization variables. For the sake of readability, we omit the details on the derivation and instead give the final result in Algorithm 1.

Algorithm 1: Extended System Level Synthesis

```
*Solve for feedforward terms* $\boldsymbol{d}_{\boldsymbol{u}}=(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}}+\boldsymbol{R})^{-1}(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{x}_d+\boldsymbol{R}\boldsymbol{u}_d)$
while $i<T$ do  *Solve for feedback terms*
    $\hat{\boldsymbol{\Phi}}_u^i=-(\boldsymbol{S}_{\boldsymbol{u}}^{i\top}\boldsymbol{Q}^i\boldsymbol{S}_{\boldsymbol{u}}^i+\boldsymbol{R}^i)^{-1}\boldsymbol{S}_{\boldsymbol{u}}^{i\top}\boldsymbol{Q}^i\boldsymbol{S}_{\boldsymbol{x}}^i$
    $\hat{\boldsymbol{\Phi}}_x^i=\boldsymbol{S}_{\boldsymbol{x}}^i+\boldsymbol{S}_{\boldsymbol{u}}^i\hat{\boldsymbol{\Phi}}_u^i$
end
*Compute the feedback and feedforward parts of the controller with*
$\boldsymbol{K}{=}\boldsymbol{\Phi}_u\boldsymbol{\Phi}_x^{-1}$ , $\boldsymbol{k}{=}(\boldsymbol{I}-\boldsymbol{K}\boldsymbol{S}_{\boldsymbol{u}})\boldsymbol{d}_{\boldsymbol{u}}$
```

4.2 Iterative system level synthesis (iSLS)

In this subsection, we consider the problem of system level synthesis for nonlinear dynamical systems and non-quadratic cost functions. We perform a first order Taylor expansion of the dynamical system $\boldsymbol{x}_{t+1}{=}\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}_t$ around some nominal realization of the plant denoted as $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t, \boldsymbol{\mu}_{w_t})$, namely, $\boldsymbol{x}_{t+1} \approx \boldsymbol{f}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \boldsymbol{\mu}_{w_t} + \boldsymbol{A}_t(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t) + \boldsymbol{B}_t(\boldsymbol{u}_t - \hat{\boldsymbol{u}}_t) + (\boldsymbol{w}_t - \boldsymbol{\mu}_{w_t})$ with Jacobian matrices $\{\boldsymbol{A}_t = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_t}|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}, \boldsymbol{B}_t = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}_t}|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}\}$. Using the notations $\hat{\boldsymbol{x}}_{t+1}{=}\boldsymbol{f}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \boldsymbol{\mu}_{w_t}$, $\Delta\boldsymbol{x}_t{=}\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t$, $\Delta\boldsymbol{u}_t{=}\boldsymbol{u}_t - \hat{\boldsymbol{u}}_t$, the linearized dynamics model can be rewritten as

$$\begin{aligned} \Delta\boldsymbol{x}_{t+1} &= \boldsymbol{A}_t \Delta\boldsymbol{x}_t + \boldsymbol{B}_t \Delta\boldsymbol{u}_t + \Delta\boldsymbol{w}_t, \forall t, \\ \iff \Delta\boldsymbol{x} &= \boldsymbol{Z}\boldsymbol{A}_d \Delta\boldsymbol{x} + \boldsymbol{Z}\boldsymbol{B}_d \Delta\boldsymbol{u} + \Delta\boldsymbol{w}, \end{aligned}$$

where the second line is the stacked form of the linearized dynamics model. We assume a controller of the form $\Delta\boldsymbol{u} = \boldsymbol{K}\Delta\boldsymbol{x} + \boldsymbol{k}$ and follow the same procedure described in the previous section to write the closed-loop responses as $\Delta\boldsymbol{x} = \boldsymbol{\Phi}_x \Delta\boldsymbol{w} + \boldsymbol{d}_x$ and $\Delta\boldsymbol{u} = \boldsymbol{\Phi}_u \Delta\boldsymbol{w} + \boldsymbol{d}_u$, where the variables $\boldsymbol{\Phi}_x, \boldsymbol{\Phi}_u, \boldsymbol{d}_x, \boldsymbol{d}_u$ satisfy the same constraints in (9) and (4) with respect to the linearized dynamics model.

For simplicity, we assume a cost function of the form $c(\boldsymbol{x}_t, \boldsymbol{u}_t){=}c(\boldsymbol{x}_t) + \|\boldsymbol{u}_t\|^2_{\boldsymbol{R}}$ where $c(\boldsymbol{x}_t)$ is potentially a non-quadratic function of the state. Then, $c(\boldsymbol{x}_t)$ can be approximated by a second order Taylor expansion around $\hat{\boldsymbol{x}}_t$, namely $c(\boldsymbol{x}_t){\approx}c(\hat{\boldsymbol{x}}_t) + \boldsymbol{c}^{\top}_{\boldsymbol{x}_t}(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t) + \frac{1}{2}(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t)^{\top}\boldsymbol{C}_{\boldsymbol{x}_t\boldsymbol{x}_t}(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t){=}\frac{1}{2}(\Delta\boldsymbol{x}_t - \boldsymbol{x}_{(d,t)})^{\top}\boldsymbol{C}_{\boldsymbol{x}_t\boldsymbol{x}_t}(\Delta\boldsymbol{x}_t - \boldsymbol{x}_{(d,t)}) + \text{const.}$, where $\boldsymbol{c}_{\boldsymbol{x}_t}{=}\frac{\partial c}{\partial \boldsymbol{x}_t}|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}$ and $\boldsymbol{C}_{\boldsymbol{x}_t\boldsymbol{x}_t}{=}\frac{\partial^2 c}{\partial \boldsymbol{x}_t^2}|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}$, and $\boldsymbol{x}_{(d,t)}{=} -\boldsymbol{C}^{-1}_{\boldsymbol{x}_t\boldsymbol{x}_t}\boldsymbol{c}_{\boldsymbol{x}_t}$. This cost can be rewritten in batch form removing the constant terms as $c(\boldsymbol{x}, \boldsymbol{u}){=}\frac{1}{2}\|\Delta\boldsymbol{x} - \boldsymbol{x}_d\|^2_{\boldsymbol{C}_{\boldsymbol{x}\boldsymbol{x}}} + \|\Delta\boldsymbol{u} - \boldsymbol{u}_d\|^2_{\boldsymbol{R}}$. We propose to perform a Newton's step at each iteration by solving a tracking problem of the same form as (9) by changing the state and the control to $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{u}$. Thus, this results in the following convex optimization problem

$$\begin{aligned} \min_{\boldsymbol{\Phi}_x, \boldsymbol{\Phi}_u, \boldsymbol{d}_x, \boldsymbol{d}_u} \quad & \|\boldsymbol{\Phi}_x \Delta\boldsymbol{w} {+} \boldsymbol{d}_{\boldsymbol{x}} {-} \boldsymbol{x}_d\|^2_{\boldsymbol{Q}} + \|\boldsymbol{\Phi}_u \Delta\boldsymbol{w} {+} \boldsymbol{d}_{\boldsymbol{u}} {-} \boldsymbol{u}_d\|^2_{\boldsymbol{R}} \\ \text{s.t.} \quad & \boldsymbol{\Phi}_x = \boldsymbol{S}_x + \boldsymbol{S}_u \boldsymbol{\Phi}_u, \\ & \boldsymbol{d}_{\boldsymbol{x}} = \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{d}_{\boldsymbol{u}}, \\ & \boldsymbol{\Phi}_x, \boldsymbol{\Phi}_u \in \mathcal{L} \end{aligned} \tag{10}$$

which can be solved analytically the same way as eSLS. Newton optimization methods are known to be prone to overshoots, which can be handled via line search. We propose a line search algorithm based on the feedforward control term of our controller as was done by previously proposed iLQR algorithms [13]. Specifically, we define a variable α with a line search strategy with $\boldsymbol{d}_u^{k+1} = \alpha \boldsymbol{d}_u^k$. We accept this update if the trajectories found by these closed loop dynamics models decrease our actual cost function, otherwise we decrease α and re-assess. This corresponds to updating $\boldsymbol{k}$ in the same manner.

4.3 Time correlations between states

The controller defined by SLS reacts not only to the current state error of the robot, but also to the history of previous states. Such a controller with mem-

Algorithm 2: Iterative System Level Synthesis (iSLS)

```
Initialize the nominal state $\hat{\boldsymbol{x}}_t$ and control $\hat{\boldsymbol{u}}_t$ ;
Initialize the change in the cost $\Delta c$ ;
Set a threshold $\tau$ ;
while $|\Delta c| > \tau$ do  Solve iSLS
    Linearize the dynamics and quadratize the cost function around
      $\{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t\}_{t=0}^{T}$ to find $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}_{\boldsymbol{xx}}$, $\boldsymbol{x}_d$ and $\boldsymbol{u}_d$ ;
    Solve (10) to find $\boldsymbol{K}$ and $\boldsymbol{k}$ ;
    Do line search to update $\boldsymbol{k}$ using the controller $\Delta\boldsymbol{u} = \boldsymbol{K}\Delta\boldsymbol{x} + \boldsymbol{k}$ and the
      dynamics model ;
    Update $\Delta c$.
end
```

ory can be exploited in tasks requiring correlations between states at different timesteps, as opposed to a controller without memory such as LQR controllers. An SLS controller can remember what it did in the previous part of the trajectory to use this information to better anticipate the future states and to successfully complete tasks where the future trajectory depends on past states. An example of such task is illustrated in Figure 1.

To achieve correlations between different timesteps, we make use of the off-block-diagonal elements of the precision matrix. Let us denote $\boldsymbol{x}_{t_1}$ and $\boldsymbol{x}_{t_2}$ the states at t_1 and t_2 that we want to correlate. The correlations that we consider in this work are in the form $\boldsymbol{C}\boldsymbol{x}_{t_1}+\boldsymbol{c}\sim\boldsymbol{x}_{t_2}$, where $\boldsymbol{C}$ and $\boldsymbol{c}$ are the coefficient matrix and the vector, respectively. We define the correlation cost $c(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2})$ as

$$\begin{aligned} c(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}) &= (\boldsymbol{C}\boldsymbol{x}_{t_1} + \boldsymbol{c} - \boldsymbol{x}_{t_2})^\top \boldsymbol{Q}_\mathrm{c} (\boldsymbol{C}\boldsymbol{x}_{t_1} + \boldsymbol{c} - \boldsymbol{x}_{t_2}), \\ &= \boldsymbol{x}_{t_1}^\top \underbrace{\boldsymbol{C}^\top \boldsymbol{Q}_\mathrm{c} \boldsymbol{C}}_{\boldsymbol{Q}_{t_1}} \boldsymbol{x}_{t_1} + (\boldsymbol{x}_{t_2} - \boldsymbol{c})^\top \underbrace{\boldsymbol{Q}_\mathrm{c}}_{\boldsymbol{Q}_{t_2}} (\boldsymbol{x}_{t_2} - \boldsymbol{c}) - 2\boldsymbol{x}_{t_1}^\top \underbrace{\boldsymbol{C}^\top \boldsymbol{Q}_\mathrm{c}}_{-\boldsymbol{Q}_{t_1 t_2}} (\boldsymbol{x}_{t_2} - \boldsymbol{c}) \\ &= \begin{bmatrix} \boldsymbol{x}_{t_1} \\ \boldsymbol{x}_{t_2} - \boldsymbol{c} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{Q}_{t_1} & \boldsymbol{Q}_{t_1 t_2} \\ \boldsymbol{Q}_{t_2 t_1} & \boldsymbol{Q}_{t_2} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{t_1} \\ \boldsymbol{x}_{t_2} - \boldsymbol{c} \end{bmatrix}. \end{aligned} \tag{11}$$

This means adding off-block-diagonal elements to a typical block-diagonal precision matrix $\boldsymbol{Q}$:=blkdiag$(\boldsymbol{Q}_0, \boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_T)$, by setting $\boldsymbol{Q}(t_1, t_1) = \boldsymbol{C}^\top \boldsymbol{Q}_\mathrm{c} \boldsymbol{C}$, $\boldsymbol{Q}(t_2, t_2) = \boldsymbol{Q}_\mathrm{c}$, $\boldsymbol{Q}(t_1, t_2) = -\boldsymbol{C}^\top \boldsymbol{Q}_\mathrm{c}$ and $\boldsymbol{Q}(t_2, t_1) = -\boldsymbol{Q}_\mathrm{c}\boldsymbol{C}$, where $\boldsymbol{Q}(t_i, t_j)$ represents the precision matrix block corresponding to the timesteps i and j.

4.4 Adaptation to new reference states

We remark that while the system responses encode the information about the precision of the task only, the feedforward part of the responses, namely, $\boldsymbol{d}_{\boldsymbol{x}}$ and $\boldsymbol{d}_{\boldsymbol{u}}$, encode the desired states $\boldsymbol{x}_d$ and the desired control commands $\boldsymbol{u}_d$. Moreover, $\boldsymbol{d}_{\boldsymbol{x}}$ and $\boldsymbol{d}_{\boldsymbol{u}}$ are a linear function of $\boldsymbol{x}_d$ and $\boldsymbol{u}_d$ (see Algorithm 1), which makes them easily recomputable when one changes the desired states and control commands while keeping the precision matrices constant. Examples include trajectory tracking with the same precision throughout the horizon, a task where the robot needs to reach different viapoints and final goal tasks. At the controller level, this only corresponds to a change in the feedforward term $\boldsymbol{k}$, while the feedback part $\boldsymbol{K}$ stays the same. We have $\boldsymbol{k} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S}_{\boldsymbol{u}})\boldsymbol{d}_{\boldsymbol{u}}, = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S}_{\boldsymbol{u}})(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R})^{-1}(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{x}_d + \boldsymbol{R}\boldsymbol{u}_d), = \boldsymbol{F}_{\boldsymbol{x}}\boldsymbol{x}_d + \boldsymbol{F}_{\boldsymbol{u}}\boldsymbol{u}_d$, where the terms $\boldsymbol{F}_{\boldsymbol{x}}{=}(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S}_{\boldsymbol{u}})(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R})^{-1}\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}$ and $\boldsymbol{F}_{\boldsymbol{u}}{=}(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S}_{\boldsymbol{u}})(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R})^{-1}\boldsymbol{R}$ can be computed offline and can be used when the desired state and control commands are changed to determine the new feedforward control term $\boldsymbol{k}$ only by matrix-vector multiplication operation along the changed time and dimensions.

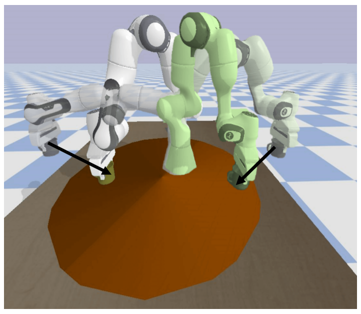
In the eSLS formulation with linear dynamics and quadratic cost, the replanned motion and the controller are optimal, i.e., if one wants to optimize from scratch when the desired state is changed, then one ends up with the same solution. However, in iSLS, the replanned motion produces valid trajectories only in the vicinity of the optimal solution. We argue and show that this can still be exploited to produce trajectories which achieve the task successfully without any computation overhead of planning from scratch.

5 Experiments and Results

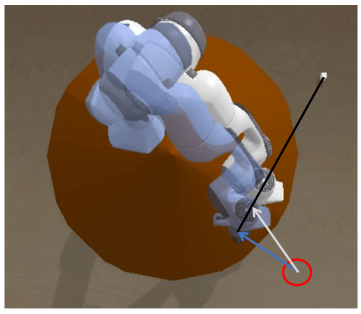
This section showcases the proposed approach with robotic tasks on a simulated and real environment with Franka Emika robot. The experiment videos for the real task can be found in the video accompanying the paper.

5.1 Simulated task

We implemented a simulated robotic task exploiting task precisions with the help of the memory feedback. The robot, initially holding a coffee mug, needs to place it on the table within a certain range depicted as brown disk shape on the table (first viapoint, t=20), pick up a sugar cube depicted as a white cube (second viapoint, t=70) and drop it onto the mug (goal point, t=100) as illustrated in Figure 1a. Depending on the initial position of the robot or the perturbations in the environment, the robot can decide on different locations to place the mug. This location needs then to be remembered to drop the sugar cube from the correct location. We implemented eSLS with a double integrator dynamics with 100 timesteps to design a feedback controller on the task space and an inverse kinematics controller to track the reference trajectory generated online by the feedback controller. The cost function have three state cost components and a control cost as $c{=}\|\boldsymbol{x}_{20} - \boldsymbol{c}_d\|^2_{\boldsymbol{Q}_{20}} + \|\boldsymbol{x}_{70} - \boldsymbol{c}_s\|^2_{\boldsymbol{Q}_{70}} + \|\boldsymbol{x}_{20} - \boldsymbol{x}_{100}\|^2_{\boldsymbol{Q}_{20,100}} + 0.01\|\boldsymbol{u}\|^2$, where $\boldsymbol{c}_d$ and $\boldsymbol{c}_s$ are the center of the brown disk and the location of the sugar cube, respectively, and $\boldsymbol{Q}_{20}{=}\,\mathrm{blkdiag}(10^3, 10^3, 10^5, 10^5, 10^5, 10^5)$, $\boldsymbol{Q}_{70}{=}10^5{\times}\boldsymbol{I}$ $\boldsymbol{Q}_{20,100}{=}\,\mathrm{diag}(10^5{\times}\boldsymbol{I}_3, \boldsymbol{0}_3)$. Figure 2a shows two examples of eSLS controller starting from two different positions and hence choosing two different locations to place the mug by anticipating that it will need to put the sugar cube inside the mug. These locations differ as there is a trade-off between the state



(a) Adaptation to two different initial positions (transparent robots) by following the arrows.



(b) Comparison of the execution of MPC-LQT (blue robot) and eSLS (white robot) controllers

Fig. 2: Simulated task where the robot, initially holding a coffee mug, needs to place it on the table within a certain range (brown disk), pick up a sugar cube (white cube) and drop it onto the mug.

and control costs. As the eSLS controller has memory, the robot can remember this location to accomplish the task.

As a baseline to compare, we chose to design an LQT controller solved by dynamic programming. Obviously, this controller has no information about the previous states and neither about the off-diagonal elements in the precision matrices, hence it can not achieve the task alone. A quick but not generalizable solution to this problem is to recompute the LQT controller after the mug is placed on the table, almost as in MPC, but recomputing the solution only once. We call this controller MPC-LQT. We argue and show that this strategy is far from the optimal behavior because it eliminates the anticipatory and memory aspects of the controller. We tested MPC-LQT against our proposed framework with 10 different initial positions of the robot end-effector, each deciding on different locations of the mug. One such execution is shown in Figure 2b, with blue and white robots illustrating MPC-LQT and eSLS strategies, respectively. Each robot, starting from the same initial position depicted by a red circle, places the mug in different locations following the blue and white arrows, and picks up the sugar cube and places it inside the mug correctly. Even though successful, one can see from the geometry in the figure that the final path taken by the MPC-LQT controller (blue) is longer than the proposed eSLS controller (white). Indeed, when we compute the costs of the tasks for both cases, we obtained a cost of 474.3±204.3 for eSLS and 1004.7±464.2 for MPC-LQT, which also quantitatively shows the better performance of the behavior produced by our proposed controller. Indeed, after the mug is placed on the table, the recomputed controller of MPC-LQT can also put the sugar cube inside the mug successfully. However, in the optimal behavior as in eSLS, the robot decides on the location of the mug by anticipating that it will need to put the sugar cube inside, which is a missing feature in the baseline.

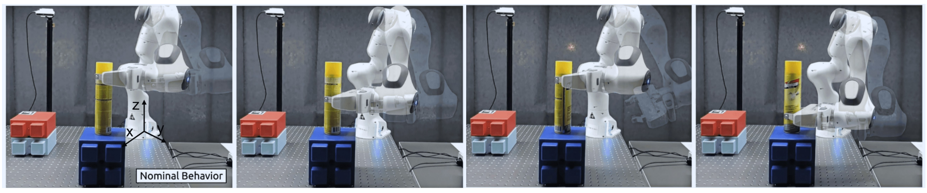


Fig. 3: Adaptation of the robot to different initial configurations shown in transparent and the grasping locations shown in solid colors.

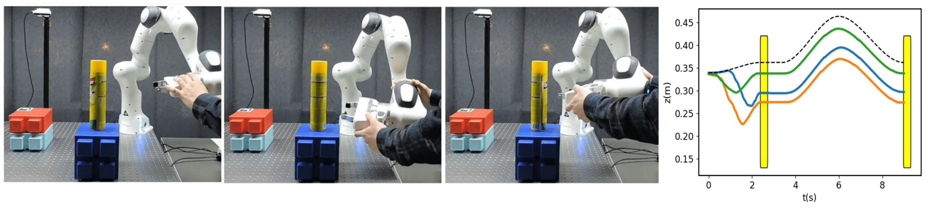


Fig. 4: Testing of the reactivity of the controller to different physical perturbations, each resulting in successful completion of the task by adaption. The plot of z-axis position (m) in time (s) gathered from the robot perturbed before grasping the object. The curves of color green, orange and blue correspond to the first, second and third screenshots respectively, while the dashed black line corresponds to the nominal solution. The robot decides on-the-fly to grasp from different locations and remembers these locations to place the yellow object without colliding with the environment.

5.2 Pick-and-place task

The robot needs to grasp a cylindrical object and place it at a given position while keeping an upright orientation of the end-effector, as seen in Figure 1. The grasping position is defined by precise x-y positions of the center of the cylinder, whereas the z-position and the rotation around the z-axis are kept at a very low precision. It results in different grasping heights that the robot needs to decide, by anticipating where it is going to place the object. This behaviour exploits the grasping affordances of a cylindrical object by allowing the robot to choose where and how it is going to grasp the object. The placing location is defined by precise x-y coordinates, while the z-coordinate is defined relatively to the z-coordinate when grasped. Consequently, the robot needs to remember where it grasped the object in order to place it at the relative placing z-position. Notice that without any memory feedback controller, the robot could try to place the object in wrong and dangerous locations as it can force the object to go downwards and collide with the environment.

Dynamics: We denote $\boldsymbol{\theta}$, $\dot{\boldsymbol{\theta}}$, $\ddot{\boldsymbol{\theta}}$, $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$, $\boldsymbol{q}$, the joint angles, the joint velocities, the joint accelerations, the end-effector positions, the end-effector velocities and the end-effector orientation in quaternion format, respectively. We define $\boldsymbol{f}^{\text{pos}}_{\text{kin}}(\cdot)$

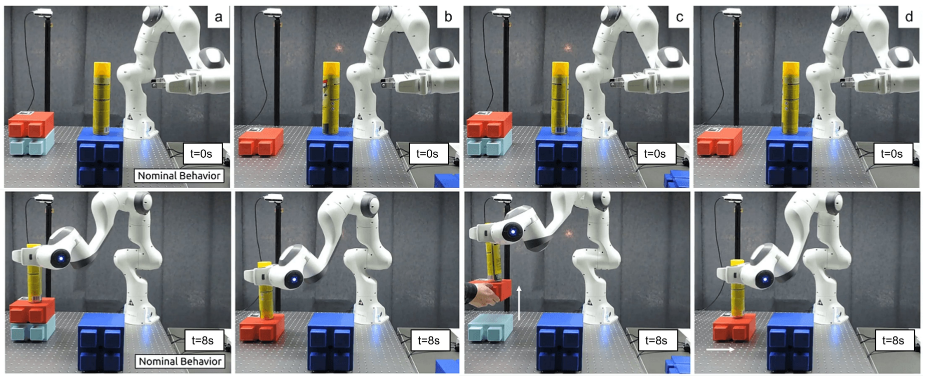


Fig. 5: (a) Nominal behavior of the robot without any adaptation. Online adaptation to a change in the final relative height (b) at $t = 0s$ (c) at $t = 3s$, and (d) at $t = 0s$ combined with an on-the-fly change in the final planar position after $t = 3s$. The robot adapts to these new goals online with a new plan according to the memory of where it grasped the object.

and $\boldsymbol{f}^{\text{orn}}_{\text{kin}}(\cdot)$ as the forward kinematics functions outputting end-effector positions and orientations, respectively. We denote $\text{Log}_{\boldsymbol{q}_0}(\boldsymbol{q}_1)$ as the logarithmic map between the quaternions $\boldsymbol{q}_0$ and $\boldsymbol{q}_1$ and define quaternion costs with the methods described in [26].

Inspired by the work in [27], we choose to incorporate the constraints and other nonlinearities into the forward dynamics model of the state in order to alleviate the problems of calculating the Hessian of the cost function. In fact, defining the forward dynamics as such allows to have a quadratic cost function with a precision matrix that defines directly the accuracy of the state itself. This, in turn, becomes very useful when one wants to replan the motion and the controller by changing not only the joint positions, but also the end-effector positions and orientations. Joint limit constraints can be handled as a soft constraint in the cost function as this approach provides very good results already without resorting to more complicated inequality constrained optimization. However, in this work, we choose to represent these limits inside the state as well to illustrate the generalization and adaptation capability of the controller for a given quadratic cost function. We first define the joint limit violation function as $\boldsymbol{f}^{\text{lim}}(\boldsymbol{\theta})=\big(\max(\boldsymbol{\theta}_l-\boldsymbol{\theta},\boldsymbol{0})+\max(\boldsymbol{\theta}-\boldsymbol{\theta}_u,\boldsymbol{0})\big)^2$, where $\boldsymbol{\theta}_l$ and $\boldsymbol{\theta}_u$ are the lower and upper bounds on the joint angles, respectively. Note that when this function is nonzero, it means that the joint limits are violated, which implies that we can use this state as quadratic function to drive it to zero. We then choose to represent the state by $\boldsymbol{z}_t=[\boldsymbol{\theta}_t,\dot{\boldsymbol{\theta}}_t,\boldsymbol{x}_t,\dot{\boldsymbol{x}}_t,\text{Log}_{\boldsymbol{q}_0}(\boldsymbol{q}_t),\boldsymbol{f}^{\text{lim}}]\in\mathbb{R}^{31}$ and the control by $\boldsymbol{u}_t=\ddot{\boldsymbol{\theta}}\in\mathbb{R}^7$ with the forward dynamics defined as $\big[\boldsymbol{\theta}_{t+1},\,\dot{\boldsymbol{\theta}}_{t+1},\,\boldsymbol{x}_{t+1},\,\dot{\boldsymbol{x}}_{t+1},\,\text{Log}_{\boldsymbol{q}_0}(\boldsymbol{q}_{t+1}),\,\boldsymbol{f}^{\text{lim}}_{t+1}\big] = \big[\boldsymbol{\theta}_t+\dot{\boldsymbol{\theta}}_t\delta t,\,\dot{\boldsymbol{\theta}}_t+\boldsymbol{u}_t\delta t,\,\boldsymbol{f}^{\text{pos}}_{\text{kin}}(\boldsymbol{\theta}_{t+1}),\,\boldsymbol{J}(\boldsymbol{\theta}_{t+1})\dot{\boldsymbol{\theta}}_t,\,\text{Log}_{\boldsymbol{q}_0}(\boldsymbol{f}^{\text{orn}}_{\text{kin}}(\boldsymbol{\theta}_{t+1})),\,\boldsymbol{f}^{\text{lim}}(\boldsymbol{\theta}_{t+1})\big]$.

Cost: The cost function is designed by three key points describing the phases of *grasping*, *lifting up* and *placing* at given timesteps. For the *grasping* phase, there is high precision on the x-y axis of the end-effector, and on the end-effector velocity, whereas the precision on the z axis is left very low to give the robot the choice to grasp anywhere along the object. The *lifting* phase is implemented so as to keep the arm safe during the pick-and-place operation to avoid any obstacle on the table. Here, we have high precision only on reaching a z-position 10 cm higher than where the object was grasped during the grasping phase, and no precision on the other dimensions. For the *placing* phase, we have high precision on the x-y positions, on keeping z position the same as where it grasped the object, and on end-effector velocity. Note that this is only possible by exploiting the off-diagonal elements in the precision matrix, as explained in Section 4.3. For the orientations, starting from the grasping phase, we set a high precision on keeping an upright orientation (but free to turn around z-axis) in order to keep the balance of the long cylindrical object.

Implementation: We implemented a 50Hz iSLS controller with a duration of 8s, which outputs the desired state and control to a 1kHz impedance controller. A vision system is implemented to track the target object location for testing the adaptation aspect of the controller to the changes in the task. We conducted several experiments with the robot to test the adaptation, reactivity and memory capabilities of the controller. We first tested the adaptation to different initial configurations. We selected randomly 3 configurations that are close to the nominal one, but still corresponding to visibly different end-effector positions and orientations, as seen in Figure 3. We noticed that the robot could adapt its grasping position to different z-dimensions because of the low precision on that axis. Here, the robot is aware of the possible grasping affordances of the object via our controller and exploits these to decide autonomously from which part of the object to grasp. It then remembers the grasping location and uses it to place the object on the desired position without colliding with the environment.

Results: For testing the reactivity to perturbations of the proposed controller, we applied some force to the robot changing its nominal behavior, i.e. different configurations corresponding to different end-effector positions and orientations, as can be seen in Figure 4. We noticed that the robot reacts to the perturbations by mostly changing its grasping locations, whilst still trying to reach the desired x-y location of the object. Even if it decides only where to grasp the object to cope with the perturbations, it can remember these locations to place the object correctly without collision as can be seen from the robot trajectories in the plot on the far right of Figure 4. In one example shown in the bottom-right of Figure 4, we changed the orientation by turning it around the z-axis. The robot's reaction to this perturbation was considerably smaller than the other perturbation types. This is indeed the expected behavior of the robot, since the error on the orientation with this change stayed very close to zero as a result of the quaternion cost function design.

Finally, we tested the proposed fast adaptation capability of the controller with memory by conducting three different experiments shown by (b), (c) and (d) in Figure 5. In all cases, we use the same optimized controller which produced the nominal behavior in (a), and reused it to replan in the other cases where the target locations are changed either at the beginning of the movement or on-the-fly. The change in the target location is tracked by the vision system following the marker on the object. In (b), we changed the final height of the placing location to a lower value at the initial time. In (c) and (d), the placing locations are changed on-the-fly by increasing the final height and changing the final x-y position, respectively. We recomputed online the feedforward part of the iSLS controller, namely $\boldsymbol{k}$ as described in Section 4.4, each time there is a significant difference between the current and detected target locations. In all cases, we see that the robot is able to adapt successfully by replanning fast to new desired states without any collision, as can also be seen in the accompanying video.

6 Conclusion

We presented an approach for synthesizing reactive and anticipatory controllers that can remember all previous states in the horizon, which is relevant for robotic tasks requiring to have a memory of previous movements. In this context, we proposed to extend SLS controllers to have a feedforward term that can handle viapoint tasks and a Newton optimization method for solving SLS for nonlinear systems and nonquadratic cost functions. We showed that our proposed method outperforms the baseline solutions for producing optimal anticipatory behaviors that require a memory feedback. We showcased our method on a high dimensional robotic system in the presence of perturbations, and demonstrated a step towards adaptation when there is a change of the desired states in the task.

SLS-based representation used in our work can facilitate the bridging between learning, planning and feedback control, especially when we do not know the cost function and/or we do not have prior knowledge on which past states should be correlated with the remaining part of the motion. This provides a very generic formulation for robot skill representations. Future work will study how to determine which past states are correlated by setting the problem as inverse optimal control. We believe that learning such correlations from demonstrations/experiences will be useful to lead the way to the development of smarter feedback controllers which can act on the memory of the states.

SLS offers the advantage that it does not require the experimenter to engineer the problem for each specific use case by augmenting the state-space with the relevant part of the history. In that sense, SLS is generic and formalizes this problem, with a homogeneous formulation, by allowing the system to freely change which past states are correlated and the way they are correlated. In this work, we tackled problems where we see important practical utility of this feature. However, instead of correlating only a couple of timesteps in this work, one could also design these tasks to remember not only one timestep but several timesteps in order to capture an important part of the movement that the robot did in the past and that it needs to remember in the future to react accordingly.

Acknowledgements

We would like to thank the reviewers for their thoughtful remarks and suggestions. This work was supported in part by the European Commission's Horizon 2020 Programme through the CoLLaboratE project (https://collaborate-project.eu/) under grant agreement 820767.

Bibliography

[1] Kamien, M.I., Schwartz, N.L.: Dynamic optimization: the calculus of variations and optimal control in economics and management. Courier Corporation (2012)

[2] Bianchi, F.D., De Battista, H., Mantz, R.J.: Wind turbine control systems: principles, modelling and gain scheduling design, vol. 19. Springer (2007)

[3] Diehl, M., Bock, H., Diedam, H., Wieber, P.B.: Fast Direct Multiple Shooting Algorithms for Optimal Robot Control, pp. 65–93. Springer Berlin Heidelberg (2006)

[4] Duchaine, V., Bouchard, S., Gosselin, C.M.: Computationally efficient predictive robot control. IEEE/ASME Transactions on Mechatronics 12(5), 570–578 (2007)

[5] Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., Hirukawa, H.: Biped walking pattern generation by using preview control of zero-moment point. In: Proc. IEEE Intl Conf. on Robotics and Automation (ICRA). vol. 2, pp. 1620–1626 (2003)

[6] Caron, S., Kheddar, A.: Multi-contact walking pattern generation based on model preview control of 3d com accelerations. In: Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids). pp. 550–557 (2016)

[7] Ponton, B., Herzog, A., Del Prete, A., Schaal, S., Righetti, L.: On time optimization of centroidal momentum dynamics. In: Proc. IEEE Intl Conf. on Robotics and Automation (ICRA). pp. 1–7 (2018)

[8] Winkler, A.W., Bellicoso, C.D., Hutter, M., Buchli, J.: Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. IEEE Robotics and Automation Letters (RA-L) 3(3), 1560–1567 (2018)

[9] Budhiraja, R., Carpentier, J., Mastalli, C., Mansard, N.: Differential dynamic programming for multi-phase rigid contact dynamics. In: Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids). pp. 1–9 (2018)

[10] Mayne, D.: Model predictive control: Recent developments and future promise. Automatica 50(12), 2967–2986 (2014)

[11] Koenemann, J., Del Prete, A., Tassa, Y., Todorov, E., Stasse, O., Bennewitz, M., Mansard, N.: Whole-body model-predictive control applied to the hrp-2 humanoid. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3346–3351 (2015)

[12] Wintz, N., Bohner, M.: Linear quadratic tracker on time scales. International Journal of Dynamical Systems and Differential Equations 3, 423–447 (2011)

[13] Li, W., Todorov, E.: Iterative linear quadratic regulator design for nonlinear biological movement systems. In: ICINCO. pp. 222–229 (2004)

[14] Kleff, S., Meduri, A., Budhiraja, R., Mansard, N., Righetti, L.: High-frequency nonlinear model predictive control of a manipulator. In: Proc. IEEE Intl Conf. on Robotics and Automation (ICRA). pp. 7330–7336 (2021)

[15] Neunert, M., Farshidian, F., Winkler, A.W., Buchli, J.: Trajectory optimization through contacts and automatic gait discovery for quadrupeds. IEEE Robotics and Automation Letters 2(3), 1502–1509 (2017)

[16] Grandia, R., Farshidian, F., Ranftl, R., Hutter, M.: Feedback mpc for torque-controlled legged robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 4730–4737 (2019)

[17] Anderson, J., Doyle, J.C., Low, S.H., Matni, N.: System level synthesis. Annual Reviews in Control 47, 364–393 (2019)

[18] Deisenroth, M.P., Neumann, G., Peters, J.: A survey on policy search for robotics. Found. Trends Robot 2(1–2), 1–142 (2013)

[19] Siekmann, J., Valluri, S., Dao, J., Bermillo, F., Duan, H., Fern, A., Hurst, J.: Learning memory-based control for human-scale bipedal locomotion. In: Proc. Robotics: Science and Systems (RSS) (2020)

[20] Zhang, M., McCarthy, Z., Finn, C., Levine, S., Abbeel, P.: Learning deep neural network policies with continuous memory states. IEEE International Conference on Robotics and Automation (ICRA) pp. 520–527 (2016)

[21] Dean, S., Tu, S., Matni, N., Recht, B.: Safely learning to control the constrained linear quadratic regulator. In: American Control Conference (ACC). pp. 5582–5588 (2019)

[22] Dean, S., Matni, N., Recht, B., Ye, V.: Robust guarantees for perception-based control. In: Proceedings of the 2nd Conference on Learning for Dynamics and Control. Proceedings of Machine Learning Research, vol. 120, pp. 350–360. PMLR (2020)

[23] Jarin-Lipschitz, L., Li, R., Nguyen, T., Kumar, V., Matni, N.: Robust, perception based control with quadrotors. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 7737–7743 (2020)

[24] Ho, D.: A system level approach to discrete-time nonlinear systems. In: American Control Conference (ACC). pp. 1625–1630 (2020)

[25] Yu, J., Ho, D.: Achieving performance and safety in large scale systems with saturation using a nonlinear system level synthesis approach. In: 2020 American Control Conference (ACC). pp. 968–973 (2020)

[26] Calinon, S.: Gaussians on riemannian manifolds : Applications for robot learning and adaptive control. IEEE Robotics and Automation Magazine 27(2), 33–45 (2020)

[27] Howell, T.A., Cleac'h, S.L., Singh, S., Florence, P., Manchester, Z., Sindhwani, V.: Trajectory optimization with optimization-based dynamics. arXiv preprint arXiv:2109.04928 (2021)