Optimal Control with State and Command Limits for a Simulated Ball Batting Task

Dennis Schüthe and Udo Frese

***Abstract*— We introduce a task oriented optimal controller based on the finite horizon quadratic regulator (LQR). The task of being at a specified position with a specified velocity in specified time is formalized by cost functions. Moreover, we include soft constraints which are part of the cost function, such that the optimal control can be computed in fixed time. We show how soft constraints help to use redundancy and point out the limitations of our soft constraint approach in simulations.**

I. INTRODUCTION

When humans are doing sports, they move fast and have to be agile. E.g. a table tennis player hits the ball in various positions of the table and always goes back to an initial position between the strikes, i.e. the middle of the table. We want to implement a similar agile task on our ball playing robot. Therefore, we developed a task level optimal controller (TLOC) whose aim is to fulfill a task in its entirety. The task is to wait in an initial position for balls thrown to the robot by an audience and to play them back. Then it should move back to its initial position after the ball has been hit. As controller we use a linear quadratic regulator (LQR) and modify it by applying appropriate time varying cost functions that modulate the task. Additionally, we add constraints on the states and the commands, to get the physical limitations of the robot included.

A. System

The system to be controlled is a three revolute joint entertainment robot, which is 2.1 m tall and plays ball games (c.f. Fig. 1). "Doggy" hits the balls with its head and tries to rebound them directly to an opponent player. The 40 cm Styrofoam head (end-effector) is connected to Axis 3 by a carbon rod. Thus, the orientation of the end-effector (EOF) does not matter. Axes 2 and 3 (pitch and roll) generate a spherical workspace (partially, due to joint limitations) on which the head can be moved. The first Axis acts like a hip (yaw) and turns the sphere around. This makes the robot redundant, as all axes intersect (c.f. Fig. 1).

The joints are driven by DC-Motors through tooth belts resulting in elasticity between joint and motor. Therefore, the controller has to take both, elasticity and redundancy, into account.

To detect the balls thrown towards Doggy, i.e. with varying speed from various positions, a stereo camera system is attached to the hip. The system is able to detect multiple balls and predict their trajectories [1]. Out of the trajectory a heuristics decides when (t_d) to place the heads center where ($\mathbf{p}_\mathrm{d}$) and with what velocity ($\mathbf{v}_\mathrm{d}$) to play the ball back. The scope of this paper is how to realize the task (t_d, $\mathbf{p}_\mathrm{d}$, $\mathbf{v}_\mathrm{d}$) by optimal control.

D. Schüthe and U. Frese are with University of Bremen, Enrique-Schmidt-Str. 5, 28359 Bremen, Germany {schuethe,ufrese}@informatik.uni-bremen.de.

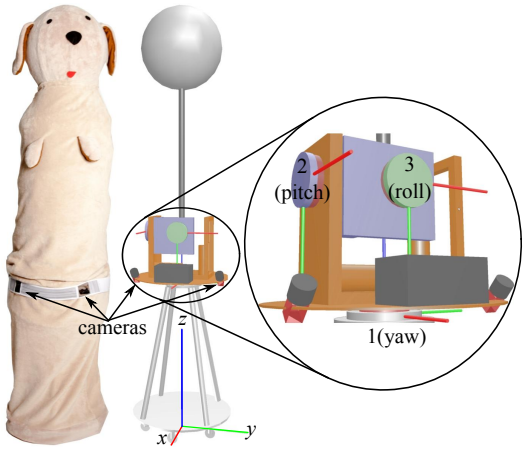

Fig. 1. Ball playing entertainment robot "Doggy" (left) and the interior as simulation (right). The colors red, green, and blue denote x, y, and z axis respectively. Numbers denote joints. The World Coordinate system is on the ground.

B. Contribution

Already, we described the task of hitting a ball back to an audience by our robot. This itself is a sporty task. We want to show that it is possible and elegant to specify this task on top of a standard optimization algorithm. Here, we want to use the optimal control algorithm of a time varying LQR as basis for our optimization. Our question is, is it effective to formulate such a task within an LQR optimization process?

We have already shown that it is elegant to describe a whole task as optimization problem [2]. Our contribution is to enhance the TLOC of [2] by soft constraints on states and commands and use them in cost functionals. Due to the optimization approach the controller then automatically utilizes the robot's redundancy to accommodate joint limits. To our knowledge the combination of LQR, model-predictive control and soft limits via cost functions into a single controller which has full freedom to fulfill a task is unique. It also shapes the motion in a way torque limits are met. Unlike trajectory planning plus position control our controller reacts to disturbances in a (linearized) optimal way.

II. RELATED WORK

The idea of TLOC is similar to [3]. They use model predictive control to get the best choice between speed

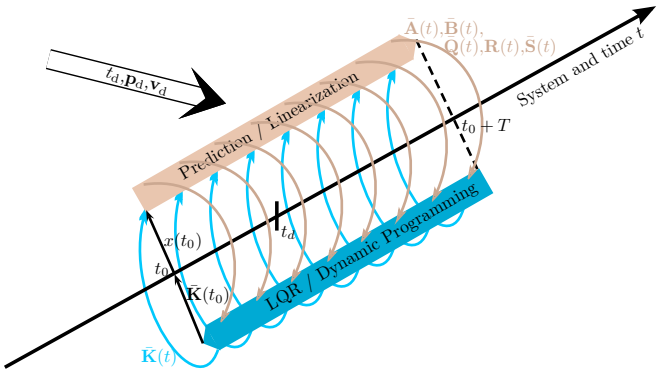

Fig. 2. Principle drawing of implemented receding horizon regulator with prediction and linearization of nonlinear dynamic system.

limitations and ramp metering on a freeway. The task is about finding the optimal traffic flow. Also, they work with a fixed horizon and predict the traffic for that, which is then used as input for the controller and optimizer. Close to our algorithm is the principle of iLQR described in [4]. Nonlinear dynamics are linearized around a trajectory which is optimized using the LQR principle iteratively. In [4] this is done offline, we perform iterations online interleaved with execution and refining goal updates provided by the tracker.

Constraints in LQR algorithms can be equality or inequality constraints. The difficulty is the number of constraints and decision variables in the optimization, which both are infinite [5]. The big advantage of soft constraints is that the solution for the optimal control is computed in fixed time and relaxes the hard constraints as shown in [6] for state constraints. Another approach on how to use constraints is given in [7] where the controller is modified to explicit suboptimal control, which reduces the horizon length as much as possible. This can not be an option for us, as our horizon length is part of the task. Another approach was given by [8], where only input constraints are used. Here, the input is basically saturated at the boundaries and by using quadratic programming algorithms the optimal control gain is calculated. This is in contrast to our goal, where we need the feedback gain and not explicitly the control. In [9] a control saturation function is handled through optimal control for LQR.

Similar batting tasks can be found in [10] and [11], where a robot is playing table tennis using movement templates and a baseball task is fulfilled using a trajectory generator respectively. Both control the direction of the rebound.

III. OVERVIEW

The task for the controller is to move the EOF such that at a given time t_d, it has a Cartesian position $\mathbf{p}_\mathrm{d}$ with velocity $\mathbf{v}_\mathrm{d}$. After the task has been defined, we want to give an overview of the implementation and the structure of this paper. Fig. 2 shows the basic concept. The algorithm iteratively optimizes the control gains $\bar{\mathbf{K}}(t)$ for every point in time by alternatingly linearizing the nonlinear dynamics (Sec. IV) and cost functions and optimally solving the linearized problem via LQR. At time t_0 we take the actual state $\mathbf{x}(t_0)$ and give it to our prediction algorithm. The prediction computes the behavior and linearizes the nonlinear system dynamics at those predicted values for a horizon T forward in time (Sec. V). The linearization of cost functions needed for the LQR are also described herein. The task of hitting the ball constitutes a special cost function at t_d, similar for returning after the hit.

The LQR uses the linearized system $(\bar{\mathbf{A}}(t), \bar{\mathbf{B}}(t))$ and cost terms $(\bar{\mathbf{Q}}(t), \mathbf{R}(t), \bar{\mathbf{S}}(t))$ to compute the optimal feedback gains $\bar{\mathbf{K}}(t)$ backward in time for each time instance of the horizon (also in Sec. V). This is done by dynamic programming. The LQR puts the feedback gain $\bar{\mathbf{K}}(t_0)$ back to the system, where it is actually used, and all gains $\bar{\mathbf{K}}(t)$ back to the prediction stage for the next iteration.

The task is described by cost functions given in Sec. VI. To handle physical limitations of the system soft constraints are introduced in Sec. VII. The whole process of linearization and optimization runs slower than the actual controller (Sec. VIII). Simulation experiments demonstrate the approach (Sec. IX).

IV. ROBOT MODEL

After a process overview in the previous section we describe the robot model our controller and the simulation is based on.

A. Kinematics

The kinematics give the world coordinate position of the EOF with respect to joint positions. It describes how joints are connected to another by rotations and translations of their coordinate systems. For our special case we do not have translations between the joints as they share a common turning point, where all axes intersect. This reduces the kinematics to rotations of axes, a translation from world coordinates to intersecting point coordinates, and a translation from intersecting point to EOF (c.f. Fig. 1).

$$f_\mathrm{kin}(\mathbf{q}) = \mathbf{T}_1^0 + \mathbf{R}_1^0(q_1)\mathbf{R}_2^1(q_2)\mathbf{R}_3^2(q_3)\mathbf{T}_{EOF}^3 \qquad (1)$$

Hereby the EOFs position is first translated to the third coordinate system with $\mathbf{T}_{EOF}^3$, then rotated with $\mathbf{R}$ and the joint's angles $\mathbf{q} = \begin{pmatrix} q_1 & q_2 & q_3 \end{pmatrix}^T$, which are equivalent for yaw, pitch, and roll respectively, down to the base coordinate system. Additionally, a translation from intersection point to ground is added. Explicitly, our robot has the kinematics

$$f_\mathrm{kin}(\mathbf{q}) = \begin{pmatrix} (\mathrm{c}_1\mathrm{s}_2\mathrm{c}_3 - \mathrm{s}_1\mathrm{s}_3)\,0.9\,\mathrm{m} \\ (\mathrm{c}_1\mathrm{s}_3 + \mathrm{s}_1\mathrm{s}_2\mathrm{c}_3)\,0.9\,\mathrm{m} \\ \mathrm{c}_2\mathrm{c}_3 0.9\,\mathrm{m} + 1\,\mathrm{m} \end{pmatrix}, \qquad (2)$$

where c_1 is $\cos q_1$, s_1 is $\sin q_1$, etc. Note that the kinematics exclude the orientation of the EOF as it is spherical.

B. Dynamics

The dynamics describe the relation between actuation and forces acting on the robot, as well as acceleration and motion that follow. Motors and joints are linked through tooth belts which have relevant elasticities. Therefore, the elastic joint

model will take the vectors of motor angles $\boldsymbol{\theta}$, velocities $\dot{\boldsymbol{\theta}}$, and accelerations $\ddot{\boldsymbol{\theta}}$ additionally to the vectors of joint angles $\mathbf{q}$, velocities $\dot{\mathbf{q}}$, and accelerations $\ddot{\mathbf{q}}$ into account [12]. The elasticity is approximately a spring of stiffness $\mathbf{K_S}$ and damping $\mathbf{D_S}$. The damping is set to zero to handle the worst case [13]. Thus, the dynamics in state space notation are

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} \dot{\mathbf{q}} \\ -\mathbf{M}_{\mathrm{J}}^{-1}(\mathbf{q})(\mathbf{c}(\mathbf{q},\dot{\mathbf{q}})+\tau_g+\mathbf{K}_{\mathrm{S}}(\mathbf{q}-\boldsymbol{\theta})) \\ \dot{\boldsymbol{\theta}} \\ -\mathbf{M}_{\mathrm{M}}^{-1}\mathbf{K}_{\mathrm{S}}(\boldsymbol{\theta}-\mathbf{q})+\mathbf{M}_{\mathrm{M}}^{-1}\boldsymbol{\tau}_m \end{pmatrix}, \quad (3)$$

with state and command vector

$$\mathbf{x} = \left(\mathbf{q}^T \ \dot{\mathbf{q}}^T \ \boldsymbol{\theta}^T \ \dot{\boldsymbol{\theta}}^T\right)^T, \quad (4)$$

$$\mathbf{u} = \boldsymbol{\tau}_m, \quad (5)$$

Where $\mathbf{M}_{\mathrm{J}}(\mathbf{q})$, $\mathbf{M}_{\mathrm{M}}$, $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$, $\tau_g(\mathbf{q})$, and $\boldsymbol{\tau}_m$ are joint inertia, motor inertia, vector of Coriolis and centrifugal terms, vector of gravitational terms and motor torque respectively. We have obtained these functions from the *Spatial_v2* library in MatLab [14], where we plugged in inertia matrices from geometries and masses from a CAD model.

V. LQR OPTIMAL CONTROL

Consider a nonlinear continuous system dynamics as in (3) and cost function for a horizon of length T starting at the current time t_0

$$J = \mathrm{cost}_{t_0+T}(\mathbf{x}(t_0+T)) + \int_{t_0}^{t_0+T} \mathrm{cost}(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau)\, d\tau, \quad (6)$$

both dependent on the state vector $\mathbf{x} \in \mathcal{X}$ of the state set and the input command $\mathbf{u} \in \mathcal{U}$ of the command set. The cost $\mathrm{cost}(\mathbf{x}(t), \mathbf{u}(t), t)$ for each time is a quadratic function

$$\begin{aligned} \mathrm{cost}(\mathbf{x}(t), \mathbf{u}(t), t) &= \sum_i w_i h_i^2(\mathbf{x}(t), \mathbf{u}(t), t) \quad (7) \\ &= h(\mathbf{x}(t), \mathbf{u}(t), t)^T \mathbf{W} h(\mathbf{x}(t), \mathbf{u}(t), t) \end{aligned}$$

with weighting $\mathbf{W} = \mathrm{diag}(w_i)$ and h a vector function that specifies how the cost depend on state and command. The optimal control is given by minimizing the cost function J with respect to $\mathbf{u}$, starting from initial state $\mathbf{x}(t_0) = \mathbf{x}_0 \in \mathcal{X}$.

A. Linearization and Discretization of Dynamics

To make use of LQR for finding the optimal control, we need a linear dynamic system

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) \quad (8)$$

with the state matrix $\mathbf{A}$ and the input matrix $\mathbf{B}$. We linearize the dynamics with a first order Taylor series around the linearization points $\mathbf{x}_p$ and $\mathbf{u}_p$, which are predicted values.

$$\begin{aligned} f(\mathbf{x}(t), \mathbf{u}(t)) \approx\ & f(\mathbf{x}_p(t), \mathbf{u}_p(t)) + \\ & \frac{\partial f(\mathbf{x}_p(t), \mathbf{u}_p(t))}{\partial \mathbf{x}}(\mathbf{x}(t) - \mathbf{x}_p(t)) + \quad (9) \\ & \frac{\partial f(\mathbf{x}_p(t), \mathbf{u}_p(t))}{\partial \mathbf{u}}(\mathbf{u}(t) - \mathbf{u}_p(t)) \end{aligned}$$

$$= \underbrace{\frac{\partial f(\mathbf{x}_p(t), \mathbf{u}_p(t))}{\partial \mathbf{x}}}_{\mathbf{A}(t)} \mathbf{x}(t) + \underbrace{\frac{\partial f(\mathbf{x}_p(t), \mathbf{u}_p(t))}{\partial \mathbf{u}}}_{\mathbf{B}(t)} \mathbf{u}(t) + \mathbf{c}(t)$$

The function $\mathbf{c}$ is

$$\begin{aligned} \mathbf{c}(t) = & f(\mathbf{x}_p(t), \mathbf{u}_p(t)) \\ & - \frac{\partial f(\mathbf{x}_p(t), \mathbf{u}_p(t))}{\partial \mathbf{x}} \mathbf{x}_p(t) - \frac{\partial f(\mathbf{x}_p(t), \mathbf{u}_p(t))}{\partial \mathbf{u}} \mathbf{u}_p(t). \end{aligned} \quad (10)$$

To fit (9) into (8), we make an affine extension to

$$\dot{\bar{\mathbf{x}}}(t) = \bar{\mathbf{A}}(t)\bar{\mathbf{x}}(t) + \bar{\mathbf{B}}(t)\mathbf{u}(t), \quad (11)$$

where $\bar{\mathbf{x}} = \left(\begin{smallmatrix} \mathbf{x} \\ 1 \end{smallmatrix}\right)$, $\bar{\mathbf{A}} = \left(\begin{smallmatrix} \mathbf{A} & \mathbf{c} \\ \mathbf{0} & 0 \end{smallmatrix}\right)$ and $\bar{\mathbf{B}} = \left(\begin{smallmatrix} \mathbf{B} \\ \mathbf{0} \end{smallmatrix}\right)$.

For discrete time steps we can discretize (11) to calculate the next state. Using linear Taylor series for the matrix exponential to solve the first order differential equation (c.f. [2], [15]), we get the discrete system

$$\bar{\mathbf{x}}_{n+1} = \underbrace{\left(\mathbf{I} + \bar{\mathbf{A}}(t)T_{\mathrm{s}}\right)}_{\bar{\mathbf{A}}_n} \bar{\mathbf{x}}_n + \underbrace{\bar{\mathbf{B}}(t)T_{\mathrm{s}}}_{\bar{\mathbf{B}}_n} \mathbf{u}_n \quad (12)$$

for the sample time T_{s}. If we get an initial state $x(0)$, we can predict forward in time, linearize, and discretize the state matrices for each step, like it is illustrated in Fig. 2. Linearizing and discretizing the state space equation (3) with (9) to (12), we can use them for the LQR as $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$.

B. Quadratization of Costs

The linearization process described in (9) can also be applied to h in (7) leading to a quadratic cost

$$\begin{aligned} h(\mathbf{x}(t), \mathbf{u}(t), t) \approx\ & h(\mathbf{x}_p, \mathbf{u}_p) + \\ & \frac{\partial h(\mathbf{x}_p, \mathbf{u}_p)}{\partial(\mathbf{x}, \mathbf{u})} \left((\mathbf{x}, \mathbf{u}) - (\mathbf{x}_p, \mathbf{u}_p)\right) \end{aligned} \quad (13)$$

$$\begin{aligned} &= \left(\begin{smallmatrix} \frac{\partial h(\mathbf{x}_p, \mathbf{u}_p)}{\partial \mathbf{x}}, & h(\mathbf{x}_p, \mathbf{u}_p) - \frac{\partial h(\mathbf{x}_p, \mathbf{u}_p)}{\partial(\mathbf{x}, \mathbf{u})}(\mathbf{x}_p, \mathbf{u}_p), & \frac{\partial h(\mathbf{x}_p, \mathbf{u}_p)}{\partial(\mathbf{u})} \end{smallmatrix}\right) \begin{pmatrix} \mathbf{x} \\ 1 \\ \mathbf{u} \end{pmatrix} \\ &= \begin{pmatrix} \bar{\mathbf{J}}_x(t) & \mathbf{J}_u(t) \end{pmatrix} \begin{pmatrix} \bar{\mathbf{x}}(t) \\ \mathbf{u}(t) \end{pmatrix} \end{aligned} \quad (14)$$

Where t is a multiple of the sample time T_{s}, so we can replace it by the discrete step n. Thus, the quadratic cost can be approximated in discrete form as

$$\begin{aligned} \mathrm{cost}_n &\approx \left(\left(\begin{smallmatrix} \bar{\mathbf{J}}_{x,n} & \mathbf{J}_{u,n} \end{smallmatrix}\right) \left(\begin{smallmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{smallmatrix}\right)\right)^T \mathbf{W} \left(\left(\begin{smallmatrix} \bar{\mathbf{J}}_{x,n} & \mathbf{J}_{u,n} \end{smallmatrix}\right) \left(\begin{smallmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{smallmatrix}\right)\right) \\ &= \left(\begin{smallmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{smallmatrix}\right)^T \left(\left(\begin{smallmatrix} \bar{\mathbf{J}}_{x,n} & \mathbf{J}_{u,n} \end{smallmatrix}\right)^T \mathbf{W} \left(\begin{smallmatrix} \bar{\mathbf{J}}_{x,n} & \mathbf{J}_{u,n} \end{smallmatrix}\right)\right) \left(\begin{smallmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{smallmatrix}\right) \\ &= \begin{pmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{pmatrix}^T \begin{pmatrix} \bar{\mathbf{Q}}_n & \bar{\mathbf{S}}_n^T \\ \bar{\mathbf{S}}_n & \mathbf{R}_n \end{pmatrix} \begin{pmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{pmatrix}. \end{aligned} \quad (15)$$

State weight $\bar{\mathbf{Q}} \succeq 0$ is symmetric positive semidefinite and command weight $\bar{\mathbf{R}} \succ 0$ is symmetric positive definite. We also have a combinatorial matrix $\bar{\mathbf{S}}$ for mixed $\bar{\mathbf{x}}, \mathbf{u}$ terms, in particular for the part linear in $\mathbf{u}$ using the 1 from $\bar{\mathbf{x}}$.

C. Algebraic Riccati Equation

We plug (15) into the overall cost functional

$$J(\mathbf{x},\mathbf{u}) = \bar{\mathbf{x}}_N^T \bar{\mathbf{Q}}_N \bar{\mathbf{x}}_N + \sum_{n=n_0}^{n_0+N-1} \bar{\mathbf{x}}_n^T \bar{\mathbf{Q}}_n \bar{\mathbf{x}}_n + 2\bar{\mathbf{x}}_n^T \bar{\mathbf{S}}_n \mathbf{u}_n + \mathbf{u}_n^T \mathbf{R}_n \mathbf{u}_n \tag{16}$$

as described in [16], [17] for the discrete finite horizon length $N = T/T_\mathrm{s}$, and starting at step $n_0 = t_0/T_\mathrm{s}$. We now derive the optimal control for step n as the one minimizing

$$J_n = \begin{pmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{pmatrix}^T \begin{pmatrix} \bar{\mathbf{Q}}_n & \bar{\mathbf{S}}_n^T \\ \bar{\mathbf{S}}_n & \mathbf{R}_n \end{pmatrix} \begin{pmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{pmatrix} + \bar{\mathbf{x}}_{n+1}^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{x}}_{n+1}. \tag{17}$$

The symmetric positive semidefinite matrix $\bar{\mathbf{P}} \succeq 0$ is the accumulated weight as solution of the discrete algebraic Riccati equation. For $\bar{\mathbf{x}}_{n+1}$ we can insert the dynamics (12), this leads to

$$J_n = \begin{pmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{pmatrix}^T \begin{pmatrix} \bar{\mathbf{Q}}_n + \bar{\mathbf{A}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{A}}_n & \bar{\mathbf{S}}_n^T + \bar{\mathbf{A}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{B}}_n \\ \bar{\mathbf{S}}_n + \bar{\mathbf{B}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{A}}_n & \mathbf{R}_n + \bar{\mathbf{B}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{B}}_n \end{pmatrix} \begin{pmatrix} \bar{\mathbf{x}}_n \\ \mathbf{u}_n \end{pmatrix}. \tag{18}$$

Minimizing (18) with respect to $\mathbf{u}$ yields the optimal control

$$\mathbf{u}_n^\star = \underbrace{-\left(\mathbf{R}_n + \bar{\mathbf{B}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{B}}_n\right)^{-1} \left(\bar{\mathbf{S}}_n + \bar{\mathbf{B}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{A}}_n\right)}_{\bar{\mathbf{K}}_n} \bar{\mathbf{x}}_n, \tag{19}$$

which defines the feedback gain matrix $\bar{\mathbf{K}}$. Plugging the result into (18) for $\mathbf{u}_n^\star$ and making use of the fact that the cost at step n is also given by $J_n = \bar{\mathbf{x}}_n^T \bar{\mathbf{P}}_n \bar{\mathbf{x}}_n$ leads to

$$\bar{\mathbf{P}}_n = \bar{\mathbf{Q}}_n + \bar{\mathbf{A}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{A}}_n + \left(\bar{\mathbf{S}}_n^T + \bar{\mathbf{A}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{B}}_n\right) \bar{\mathbf{K}}_n. \tag{20}$$

We see that $\bar{\mathbf{P}}$ and $\bar{\mathbf{K}}$ can be computed recursively. For the last step $n = N$ the weight is $\bar{\mathbf{P}}_N = \bar{\mathbf{Q}}_N$. Moreover, we would like to mention that in the case of $\bar{\mathbf{S}} = \mathbf{0}$, (19) and (20) go over to standard LQR equations.

VI. TASK LEVEL OPTIMAL CONTROL

So far we showed optimal control with the iterated LQR and described the kinematics and dynamics for our robotic system. Now, we define the task fulfilled by the controller and hence the cost functions to achieve this task.

The task can be divided into two jobs. First, waiting for a ball intersecting with the workspace and moving towards the intersection point (desired position) $\mathbf{p}_\mathrm{d}$ to hit the ball at the right time t_d, with velocity $\mathbf{v}_\mathrm{d}$. This needs to be very precise, as our goal is to hit the ball back to an opponent player. The second job is to move the robots head back to its initial upright position and wait for new balls. The back movement finished at a heuristically given time t_r. Both jobs are combined to one task of discrete horizon N and mapped to discrete cost functions. This gives whole controllability to the controller to decide the best way to reach the goal and still react on disturbances from outside. Planning beyond t_d prevents reaching an unrecoverable state, e.g. regarding joint limits, at t_d. For better readability all costs are given as continuous functions, using (7) and (13) to (15) to get the discrete costs, w denotes the weight.

Cost functions penalizes the state and the control. We can use them to get the state to a desired state $\mathbf{x}_\mathrm{d}$ or to stay at that state. For both jobs of the task, i.e. go and hit the ball or go to initial position, we insert a vibration reduction.

A. Vibration Reduction

Due to the elasticity between motor and joint, the system oscillates in the spring $(\mathbf{q} - \boldsymbol{\theta})$. We avoid resonance by penalizing spring motions with a vibration cost

$$\mathrm{cost}_\mathrm{vib}(\mathbf{x},t) = w_\mathrm{vib} \left(\dot{\mathbf{q}}(t) - \dot{\boldsymbol{\theta}}(t)\right)^2. \tag{21}$$

B. Desired Cartesian Position

To hit the ball we define a cost function that penalizes a deviation from the desired Cartesian position at *that time*. The state can be translated to a Cartesian position by the kinematics (2). Thus, the position cost is

$$\mathrm{cost}_\mathrm{p}(\mathbf{x},t) = \begin{cases} 0 & \forall t \neq t_\mathrm{d} \\ w_\mathrm{p} \left(f_\mathrm{kin}(\mathbf{q}(t_\mathrm{d})) - \mathbf{p}_\mathrm{d}\right)^2 & \text{otherwise} \end{cases}. \tag{22}$$

C. Desired Cartesian Velocity

The actual velocity can be computed by the state $\mathbf{x}$ and the kinematics as

$$f_\mathrm{vel}(\mathbf{x}) = \frac{\partial f_{kin}(\mathbf{q})}{\partial t} = \frac{\partial f_\mathrm{kin}(\mathbf{q})}{\partial \mathbf{q}} \dot{\mathbf{q}}. \tag{23}$$

We can penalize the desired Cartesian velocity equally to the Cartesian position with

$$\mathrm{cost}_\mathrm{v}(\mathbf{x},t) = \begin{cases} 0 & \forall t \neq t_\mathrm{d} \\ w_\mathrm{v} \left(f_\mathrm{vel}(\mathbf{x}(t_\mathrm{d})) - \mathbf{v}_\mathrm{d}\right)^2 & \text{otherwise} \end{cases}. \tag{24}$$

D. Return to Initial Position

To accomplish the task after hitting the ball, we need to return the robot's head to its initial position, i.e. $\mathbf{q}_\mathrm{r} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$ and $\dot{\mathbf{q}}_\mathrm{r} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$ m/s. Hence, we penalize deviations of position and velocity to its initials by

$$\mathrm{cost}_r(\mathbf{x},t) = \begin{cases} 0 & \text{for } t < t_\mathrm{r} < t_\mathrm{e} \\ w_\mathrm{pr}\left(\mathbf{q}^2(t) + \frac{1}{\mathrm{s}}\dot{\mathbf{q}}^2(t)\right) & \text{for } t = t_\mathrm{e} < t_\mathrm{r} \\ w_\mathrm{pr}\mathbf{q}^2(t) + w_\mathrm{vr}\dot{\mathbf{q}}^2(t) & \text{for } t_\mathrm{r} \leq t \leq t_\mathrm{e} \end{cases} \tag{25}$$

where $t_\mathrm{e} = t_0 + T$. The second condition is an advanced notice for the controller instructing it that it has to go back to initial state eventually. Here we use the same weighting factor for position and velocity. The last condition enforces home position and velocity after t_r.

E. Motor Torque

Our goal is to hit a ball with lowest energy needed for the rebound. This leads to a cost function that penalizes the input torque for minimizing heat dissipation of the DC-motor, and evenly distributes it over time. So we have

$$\mathrm{cost}_\tau(\mathbf{u},t) = \mathbf{u}^T(t) w_\mathrm{u} \mathbf{u}(t). \tag{26}$$

VII. CONSTRAINTS

By now our controller is able to handle the task using the predefined cost functions. But it does not know any restrictions given by the physical system. Now we specify soft constraints using cost functions. First, we consider the restrictions given by the robot itself, i.e. joint limitations. Afterwards, we get to limitations given by the motor.

A. State Constraints

We want the state $\mathbf{x}$ to remain in the constrained state set $\mathbb{X} \subset \mathcal{X}$ for all time steps n. This could be done by extending the quadratic cost function $\bar{\mathbf{x}}_n^T \bar{\mathbf{P}}_n \bar{\mathbf{x}}_n$ in (20) by a linearized state constraint matrix $\mathbf{H}$ such that $\mathbf{H}\mathbf{x} \preceq \mathbf{l}$, where $\mathbf{l}$ is the constraint level, as in [5]. However, while $\mathbf{P}$ has constant size, $\mathbf{H}$ may grow with the number of steps.

Instead the basic LQR algorithm is adapted to fulfill the constraints, by adding a barrier function into the cost such that costs are increasing, the nearer the state is to constraints. Therefore, the limitation or constraint cost is

$$\mathrm{cost}_{\mathrm{lim}}(\mathbf{x}, t) = h_{\mathrm{lim}}^T(\mathbf{x}, t) w_{\mathrm{lim}} h_{\mathrm{lim}}(\mathbf{x}, t), \tag{27}$$

$$h_{\mathrm{lim}}(\mathbf{x}, t) = \left[\mathrm{diag}(\mathbf{q}_{\mathrm{lim}})^{-1} \left(\mathbf{q}(t) - \mathbf{q}_{\mathrm{c}}\right)\right]^{\gamma_x}, \tag{28}$$

where $\mathbf{q}_{\mathrm{lim}}$ is the constrained limitation, $\mathbf{q}_{\mathrm{c}}$ is the center point of the lower and upper boundary limit – here a zero vector, and γ_x influences the steepness of the cost. The higher γ_x is chosen, the more the cost converges to a hard barrier, increasing rapidly beyond the limitation point $\mathbf{q}_{\mathrm{lim}}$.

Summing up all state costs yields

$$\begin{aligned}\mathrm{cost}(\bar{\mathbf{x}}, t) = \mathrm{cost}_{\mathrm{vib}}(\bar{\mathbf{x}}, t) + \mathrm{cost}_{\mathrm{lim}}(\bar{\mathbf{x}}, t) \\ + \mathrm{cost}_{\mathrm{r}}(\bar{\mathbf{x}}, t) + \mathrm{cost}_{\mathrm{v}}(\bar{\mathbf{x}}, t) + \mathrm{cost}_{\mathrm{p}}(\bar{\mathbf{x}}, t).\end{aligned} \tag{29}$$

B. Input Constraints

Another physical limitation regards the commanded motor torques for the DC-motors. Hence, we have to fulfill that the control input $\mathbf{u}$ is in set $\mathbb{U} \subset \mathcal{U}$. For command constraints we can do the same implementation as for states, i.e. a barrier function for commands

$$\mathrm{cost}_{\mathrm{ulim}}(\mathbf{u}, t) = h_{\mathrm{ulim}}^T(\mathbf{u}, t) w_{\mathrm{u}} \alpha h_{\mathrm{ulim}}(\mathbf{u}, t), \tag{30}$$

$$h_{\mathrm{ulim}}(\mathbf{u}, t) = \left(\mathrm{diag}(\mathbf{u}_{\mathrm{lim}})^{-1} \mathbf{u}(t)\right)^{\gamma_u}, \tag{31}$$

with α, γ_u, and $\mathbf{u}_{\mathrm{lim}}$ the penalization of exceeding the barrier, the gradient influence on the barrier function, and the upper and lower bound of constraints on the command respectively. Beyond the command limitation the costs are increasing rapidly. As a remark (30) leads to a nonlinear h-entry in (7) which in turn leads to a constant term in the linearized h in (14) and a $\mathbf{u}$-linear term in (15) which requires the $\mathbf{S}$ part of (15) as it binds to the 1 in $\bar{\mathbf{x}}$.

C. Damping

The price for not having constraints in the LQR-core is that the barrier functions make the costs much more nonlinear. We experienced that this leads to oscillations over the nonlinear iterations (Fig. 2). We therefore penalize control change *between iterations* by adding a damping term

$$\mathrm{cost}_{\mathrm{dmp}}(\mathbf{u}, t) = w_{\mathrm{u}} \delta \left(\mathbf{u}(t) - \mathbf{u}_p(t)\right)^2, \tag{32}$$

Algorithm 1: Infinite Task on PC side.

```
get initial state x(0) from microcontroller
initialize Ā, B̄ around linearization point x(0)
Set K̄ = 0
while ∞ do
    wait for camera update
    remove first N_up elements of K̄ and append N_up times K̄_{N-1}
    update state x(0) from microcontroller
    compute desired head center t_d, v_d, p_d from ball trajectory
    predict system behavior [Ā, B̄, x̄_p, u_p] using Eq. (3) and (9)–(12)
    quadratize cost Q̄, R̄, S̄ for x̄_p, u_p with Eq. (21)–(32), (13)–(15)
    compute gain K̄ with Eq. (19) and (20)
    transfer gain matrix to microcontroller
end
```

where δ is the damping factor. Note that this does not change the optimum itself, because after convergence $\mathbf{u} = \mathbf{u}_p$, but reduces the change in each iteration helping convergence, similar to the Levenberg-Marquardt method [18].

Adding up all command costs gives

$$\mathrm{cost}(\mathbf{u}, t) = \mathrm{cost}_{\tau}(\mathbf{u}, t) + \mathrm{cost}_{\mathrm{lim}}(\mathbf{u}, t) + \mathrm{cost}_{\mathrm{dmp}}(\mathbf{u}, t). \tag{33}$$

VIII. CHAINED IMPLEMENTATION

The controller consists of two distributed stages, a nonlinear stage and a linear stage. The linear stage shall be running on a microcontroller with a sampling rate of 1 kHz. It updates the actual state $\mathbf{x}_n$ and computes the optimal control value $\mathbf{u}_n = \bar{\mathbf{K}}_n \bar{\mathbf{x}}_n$ using the feedback gain matrix, which was computed at the nonlinear stage of the controller.

The nonlinear stage runs on a computer with a much lower update rate, of 50 Hz, due to the camera update, where the ball detection takes place. Thus, we use the new information of the ball tracking and put that information into our controller, together with the actual system state $\mathbf{x}$ given by the microcontroller. To get the new feedback gain, we first predict the system states for the horizon length N together with the gain values from the previous iteration. The system is then linearized by (9) and discretized into the time-variant system (12). The feedback gain results from a backward recursion from horizon length $N + n_0$ down to $n = n_0$ using (20) and (19). The new feedback gain is transferred to the microcontroller for linear control. This iteration is summarized in Algorithm 1 and in Fig. 2. Every 20 ms one iteration of the linearized optimization is conducted, so the process of convergence to the nonlinear optimum runs parallel to the updates by the tracker and the robot's motion.

This scheme has desirable properties: The controller reacts immediately (1 ms) to disturbances in a linearized optimal way, because an optimal $\mathbf{K}_n$ not an optimal $\mathbf{u}_n$ has been computed.

TABLE I

COST FUNCTION PARAMETERS USED IN THE SIMULATIONS.

Factor	Value	Factor	Value	Factor	Value
w_v	$5 \times 10^6 \frac{s^2}{rad^2}$	w_p	$5 \times 10^7 \frac{1}{rad^2}$	N	1000
w_vr	$1 \frac{s^2}{rad^2}$	w_pr	$50 \frac{1}{rad^2}$	N_up	20
w_vib	$10 \frac{s^2}{rad^2}$	w_lim	$20 \frac{1}{rad^2}$	γ_x	8
$\mathbf{q}_\mathrm{lim}$	$\begin{pmatrix} 170 \\ 40 \\ 30 \end{pmatrix}$ deg	α	5	δ	100
w_u	$0.01 \frac{1}{(\mathrm{Nm})^2}$	$\mathbf{u}_\mathrm{lim}$	$\begin{pmatrix} 20 \\ 20 \\ 20 \end{pmatrix}$ Nm	γ_u	1

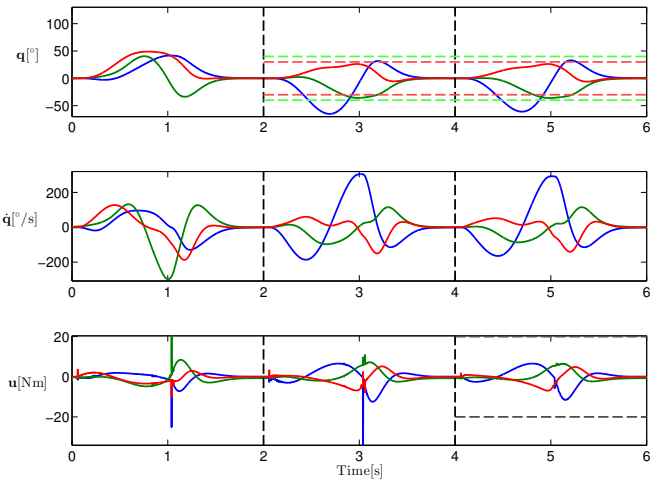

Fig. 3. Comparison between constraints inactive (left), constraints on states active (middle), and constraints active (right). Dashed lines show the limits. Blue, green, and red denote yaw, pitch, and roll axis respectively in joint position, joint velocity and command from top to bottom.

IX. SIMULATION EXPERIMENTS

In this section we evaluate the behavior of our algorithm and show the principle of soft constraints in few examples. We obtained the weights heuristically by observation of each cost and choosing the weight so costs are comparable (Tab. I). The horizon length is chosen with respect to ball flight times which are between $0.4\,\mathrm{s}$ and $0.9\,\mathrm{s}$ and we gave a little offset to it to have the return to the initial position in mind.

A. Constraint Behavior

First, we compare the controller to the same controller with no constraint or partial constraints activated. Fig. 3 shows this behavior. The desired position $\mathbf{p}_\mathrm{d} = \begin{pmatrix} -0.4762 & 0.3813 & 1.6617 \end{pmatrix}^T$ m and the velocity $\mathbf{v}_\mathrm{d} = \begin{pmatrix} -1.9868 & -3.0933 & 0.3526 \end{pmatrix}^T$ m/s have to be reached at the times $t_\mathrm{d} = [1.025, 3.025, 5.025]\,\mathrm{s}$. We have chosen this position as it is only reachable by turning the first Axis.

The left region shows inactive constraints and joint angles above the physical limit. Without constraints the controller can move Axis two and three to all points on the sphere, thus Axis three goes above its limit. Additionally, we see two commands exceed the limits during return motion. The peak is as the spring between joint and motor has to be tightened so that the joint follows the motor.

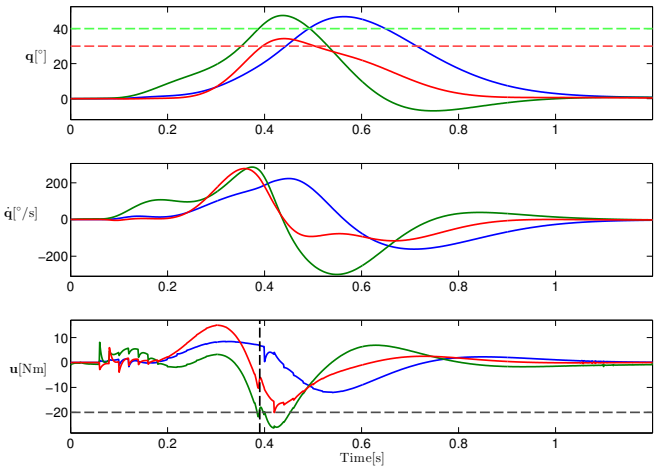

Fig. 4. Limitations of soft constraint approach. The desired position is at the edge of the reachable state space $\mathbb{X}$ with high velocity pointing towards the outside. In an attempt to fulfill the impossible task the controller exceeds the soft constraints. Horizontal dashed lines are limitations, vertical dashed line denotes the desired time. Blue, green, and red denote yaw, pitch, and roll axis respectively in joint position, joint velocity and command from top to bottom.

In the center region we activated the state constraints. The controller chooses another way to reach the goal position meeting joint limits. Moreover, we see a use of the redundant yaw Axis one, as this is the only choice to make to stay in the constraint set. Therefore, it actively uses the redundant Axis one, which in the left region is only moved passively with ≈ 0 torque. Still a hard peak is seen at the command which exceeds the limits too.

The right region shows active constraints on states and commands. The commands and the joint angles stay in their constraint set. Axes two and three are hard at the limit of their constraints. Moreover, the commands limits are clearly met. Note that these constraints need the damping (32) to converge.

B. Limitations of Soft Constraints

In the next example we show the limitations of the soft constraint approach. The desired position is $\mathbf{p}_\mathrm{d} = \begin{pmatrix} 0.3447 & 0.5785 & 1.5971 \end{pmatrix}^T$ m with the velocity $\mathbf{v}_\mathrm{d} = \begin{pmatrix} -1.2284 & 4.6929 & -3.8378 \end{pmatrix}^T$ m/s. Both are on the edge of the constraint set and the desired time $t_\mathrm{d} = 0.39\,\mathrm{s}$ to reach the goal is very short, such that the controller needs to adapt fast to the situation and act hard (c.f. Fig. 4). Moreover, the controller tries to fulfill this impossible task. From time $0.1\,\mathrm{s}$ to $0.2\,\mathrm{s}$ the controller engages and the nonlinear iterations are visible in commands discontinuities every $N_\mathrm{up} = 20\,\mathrm{ms}$, which are caused by discontinuities of the gains after an iteration. Afterwards, the strategy to get to the goal is found. Here, the soft constraints get visible. The controller has to find a compromise between cost of reaching the goal and fulfilling the constraints. After the targeted time, costs for desired state are off, so the controller violates the command constraints to get back fast into the state constraint set. Both violations are explained by the *soft* limit we have. The

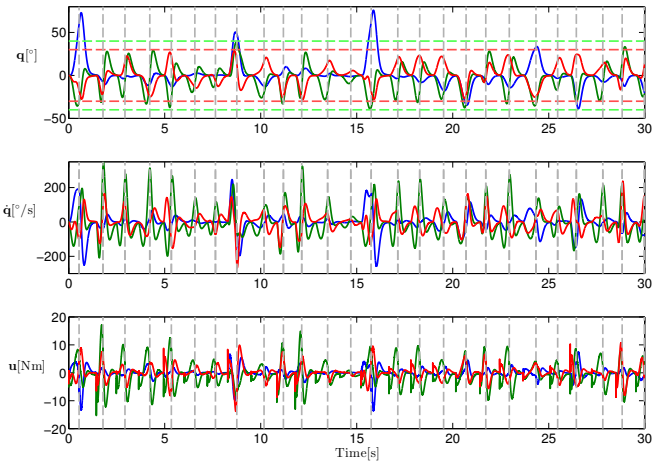

Fig. 5. Example of intended use. Balls should be hit to the front of the robot. The supplemented video moves the robot with this behavior and shows the accuracy of getting to a position with desired speed at desired times (dashed vertical lines). Note how the system actively uses redundancy in yaw Axis 1 to avoid joint limits of Axes 2 and 3. Blue, green, and red denote yaw, pitch, and roll axis respectively in joint position, joint velocity and command from top to bottom.

rationale is that soft limits need some margin to hard limits and the latter need to be monitored to abandon motion in this case. Such a margin is, however, actually desirable to be able to react on disturbances.

C. Example of Intended Ball Playing

Finally, we show how the robot acts when playing ball games. Therefore, a sequence of desired positions, velocities and times roughly corresponding to balls from positive x are chosen. The result is shown in Fig. 5 and in the supplemented video. Note that the head is removed in the video as the position and velocity are given for the center of the head from precomputation. It can be seen, that the goal is reached accurately and the desired velocity vector (marked as green arrow with fixed length) fits to the rod movement. Moreover we would argue, that the motion shows "sporty elegance" and this is because it is obtained from an optimization.

X. CONCLUSION AND FUTURE WORK

We have shown an LQR algorithm with soft constraints given by cost functionals. This brings the advantage of finite and fixed computational times. This approach works on nonlinear dynamics and non quadratic-cost functions. However, the solution entails the linearization of the dynamics – which includes an affine extension of the LQR– and the quadratization of the cost functions. Moreover, the TLOC is able to intelligently distribute commands over the axes by making use of the redundant degree of freedom, which is a result of the soft constraint approach. Restrictions of the approach were shown in simulations.

Future work is to bring the controller to the real robot. This includes monitoring hard limits, calibrating system properties and developing an estimator for $\mathbf{q}$ and $\dot{\mathbf{q}}$ from a head mounted IMU.

ACKNOWLEDGMENT

This work has been supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

REFERENCES

[1] O. Birbach and U. Frese, "A precise tracking algorithm based on raw detector responses and a physical motion model," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2013), Karlsruhe, Germany*, 2013, pp. 4746–4751.

[2] D. Schüthe and U. Frese, "Task level optimal control of a simulated ball batting robot," in *ICINCO 2014 11th International Conference on Informatics in Control, Automation and Robotics*, J. Filipe, O. Gusikhin, K. Madani, and J. Sasiadek, Eds., vol. 2. SCITEPRESS, 2014, pp. 45–56.

[3] A. Hegyi, B. De Schutter, and H. Hellendoorn, "Model predictive control for optimal coordination of ramp metering and variable speed limits," *Transportation Research Part C: Emerging Technologies*, vol. 13, no. 3, pp. 185–209, June 2005.

[4] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *ICINCO (1)*, 2004, pp. 222–229.

[5] P. Scokaert and J. Rawlings, "Constrained linear quadratic regulation," *Automatic Control, IEEE Transactions on*, vol. 43, no. 8, pp. 1163–1169, Aug 1998.

[6] M. Zeilinger, M. Morari, and C. Jones, "Soft constrained model predictive control with robust stability guarantees," *Automatic Control, IEEE Transactions on*, vol. 59, no. 5, pp. 1190–1202, May 2014.

[7] T. A. Johansen, I. Petersen, and O. Slupphaug, "Explicit sub-optimal linear quadratic regulation with state and input constraints," *Automatica*, vol. 38, no. 7, pp. 1099–1111, 2002.

[8] J. B. Mare and J. A. D. Don, "Solution of the input-constrained lqr problem using dynamic programming," *Systems & Control Letters*, vol. 56, no. 5, pp. 342 – 348, 2007.

[9] R. Goebel, "Stabilizing a linear system with Saturation Through optimal control," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 650–655, May 2005.

[10] J. Kober, K. Mulling, O. Kromer, C. H. Lampert, B. Scholkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 853–858.

[11] T. Senoo, A. Namiki, and M. Ishikawa, "Ball control in high-speed batting motion using hybrid trajectory generator," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, May 2006, pp. 1762–1767.

[12] A. Albu-Schäffer, C. Ott, and G. Hirzinger, "A unified passivity-based control framework for position, torque and impedance control of flexible joint robots," *The International Journal of Robotics Research*, vol. 26, no. 1, pp. 23–39, 2007.

[13] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2008.

[14] R. Featherstone, "Spatial_v2 (version 2)," June 2012. [Online]. Available: http://royfeatherstone.org/spatial/v2/notice.html

[15] J. Berg, S. Patil, R. Alterovitz, P. Abbeel, and K. Goldberg, "Lqg-based planning, sensing, and control of steerable needles," in *Algorithmic Foundations of Robotics IX*, ser. Springer Tracts in Advanced Robotics, D. Hsu, V. Isler, J.-C. Latombe, and M. Lin, Eds. Springer Berlin Heidelberg, 2011, vol. 68, pp. 373–389.

[16] J. C. Willems, "Least squares stationary optimal control and the algebraic Riccati equation," *Automatic Control, IEEE Transactions on*, vol. 16, no. 6, pp. 621–634, 1971.

[17] B. D. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*. Courier Corporation, 2007.

[18] W. H. Press, S. A. Teukolsky, W. T. Vettering, and B. P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing (2nd edn)*. Cambridge UP, 2002.