Task Level Optimal Control of a Simulated Ball Batting Robot

Dennis Schüthe and Udo Frese
Multi-Sensor Interactive Systems Group, University of Bremen, Enrique-Schmidt-Str. 5, Bremen, Germany

Keywords: Optimal Control, Finite Horizon, LQR, Affine System, Entertainment Robot, Redundancy, Flexible Joints.

Abstract: We developed a task-oriented controller based on optimal finite horizon control. We demonstrate this on a flexible ball playing robot with redundant degrees of freedom. The task is to reach a specified Cartesian position and velocity of the bat at a specified time, in order to rebound the ball. The controller must maintain high accuracy and react to disturbances and changing conditions. Therefore, we formulate this as an optimal control problem giving the controller the possibility to autonomously distribute motor torques amongst the redundant degrees of freedom. In simulations, we show the accuracy of the controller, the intelligent distribution of motor torques, as well as robustness against disturbances and adaptation to changing conditions.

1 INTRODUCTION

Targeted ball hitting poses a great challenge to the human bio-physical system. From perception of the ball to the exact hitting-time many steps have to be processed. Herein the "control" of muscles to trigger the bat precisely to the predicted position with appropriate speed has a crucial role in playing the ball back. In performing this action, it would seem that the human brain solves an optimal control problem. Our goal is to achieve something similar on a ball playing entertainment robot (Figure 1). Therefore, we developed an optimal finite horizon controller handling the task of being at a specified time in a specified position with a specified velocity.

1.1 System

Our robot is a 2.1 m tall ball playing entertainment robot called "Doggy". It hits balls with its head, which consists of a 40 cm styrofoam sphere fitted to a carbon rod. The rod is attached to three revolute joints having one common point where their rotational axes intersect (Figure 2). The first axis is acting like a hip, giving the robot a redundant degree of freedom (DOF) with the intention to make it more agile. Thus, the end-effector (EOF) moves on a sphere (actually partially due to joint limits) and we have three redundant joints to control the 2 DOF. As the end-effector (bat) is a sphere the orientation does not matter. Furthermore, our stereo camera is mounted after the first axis allowing it to turn and change the viewing direction.

Figure 1: The 2.1 m tall ball playing entertainment robot "Doggy".

The joints are driven by DC motors through tooth belts, resulting in elasticity, which needs to be taken into account by the controller.

1.2 Problem Statement

The problem is to be at a specified time T in a specified position $\mathbf{p}$ with a specified velocity $\mathbf{v}$ with the centre of the robots head to play back a thrown ball. The cost of doing so should be brought to a minimum, i. e. the reduction of the motors torque. Thus, the re-

Schüthe D. and Frese U..
Task Level Optimal Control of a Simulated Ball Batting Robot.
DOI: 10.5220/0005026100450056
In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2014), pages 45-56
ISBN: 978-989-758-040-6

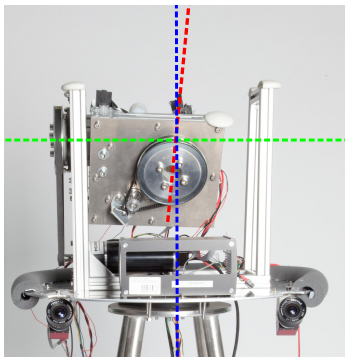

Figure 2: Interior of "Doggy" consisting of three revolute joints. Their rotational axes intersect in one point leading to redundancy. The bat, consisting of a sphere (Doggy's head) attached to a rod, is not shown.

dundancy should be intelligently used to distribute the task to all joints. Moreover, the position, velocity and time specification is needed for a controlled play back game and the robot has to comply to all three accurately. We want a single controller fulfilling this task of trajectory planning and intelligent distribution of motor torques and joint angles to reach the goal position and velocity in time. Thus, we describe this process as task level optimal control (TLOC).

Figure 3 shows the basic idea of the controllers implementation on the system. As input parameters the controller uses the actual state vector $\mathbf{x}$ consisting of velocities and positions of motors and joints. Moreover, it needs the goal parameters T, $\mathbf{p}$ and $\mathbf{v}$ from a ball tracking algorithm to compute the optimal policy. The policy is then combined with the state vector and results to a torque control vector $\mathbf{u}$. We want to clarify that the detection of the goal parameters as well as the tracking is not part of this paper.

1.3 Contribution

We combine trajectory planning and controlling in a single controller using finite horizon optimal control. The point we want to make in this paper, is that for this task or similar tasks involving something to be fulfilled at a given time, it is both elegant and practical to formulate the task as an optimal control problem. In particular the approach has the following benefits:

- Automatic utilisation of redundancy
- Automatic generation of optimal motion
- Appropriate reaction on disturbances depending on time (unlike trajectory planning plus position control)
- Computation of an optimal (time dependent) closed loop gain matrix (as opposed to an optimal trajectory or optimal commands)

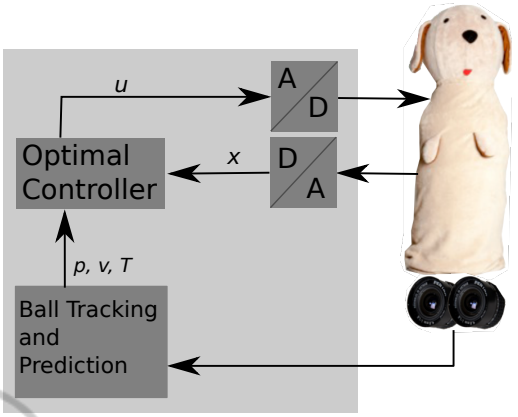

Figure 3: System overview of the controllers task bringing the robots head in time T to a specified position $\mathbf{p}$ with velocity $\mathbf{v}$. The controller provides an optimal torque $\mathbf{u}$ to fulfil this task. In Section 5.6 we will reconfigure this layout to a chained implementation.

2 RELATED WORK

Most recently the motion planning described in (Goretkin et al., 2013) is strongly related to our approach; although both have been independently developed. A finite time linear quadratic regulator (LQR) is used together with RRT* to find optimal solutions to planning problems. We have in common the finite horizon LQR, where the time is an additional input parameter. The difference is that we provide an independent feedback controller gain, whereas they provide a trajectory to follow. The best trajectory is found by iterate through all possible trajectories. This is time consuming, which is neither applicable nor necessary for the ball batting task as it does not include obstacles.

In (Perez et al., 2012) the feedback gain is computed additionally to the cost matrix, but not explicitly taken as controller gain. (Reist and Tedrake, 2010) provides a control-policy look up table of feedback stabilised trajectories, which all lead to the goal state, by simulation-based LQR-Trees.

A lot of work concerning ball hitting tasks focuses on learning strategies (Kober et al., 2010; Muelling et al., 2013), where basic movements are learned by teach in and later adapted to the situation, or are based on trajectory planning in conjunction with a joint controller to follow them as described in (Senoo et al., 2006; Hu et al., 2010; Zhang and L, 2007; Nakai et al., 1998).

Except for the volleyball robot of (Nakai et al., 1998), all have the camera system separated from the robot and they use industrial robots, which provide a position interface, such that the focus is more on the

trajectory planning than on the controlling itself. The volleyball robot is more designed as an entertainment robot focusing on sport interaction playing balls back to the opponent. But it is fixed to a specific court in a hall, which provides constant conditions leading to a simplified controller.

The previous version of our robot (Laue et al., 2013) is not fixed to a dedicated environment and is able to hit balls by moving to the predicted position, but not with a specified velocity, so it can only intercept and not play back.

In the following we describe the robot model with its task space and limitations. Furthermore, we explain the dynamic programming procedure for the optimal control process, followed by simulation experiments. At the end we give a conclusion and an overview about our future work.

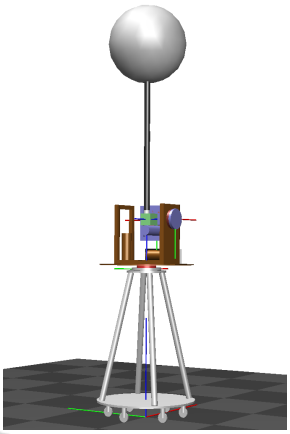

Figure 4: "Doggy" kinematics and dynamics model with coordinate systems, where red, green, blue is the x, y and z coordinate respectively. The world coordinate system is on the ground in the centre of the robot.

3 MODEL

After giving an overview of the task, to rebound a ball by controlling the EOF position such that it reaches a goal position and velocity, we describe the model of the robot our simulation and control algorithm is based on. First rigid body kinematics and dynamics are provided and later extended to the flexible joint case. We also mention the restrictions made.

3.1 Kinematics

The Kinematics describes the EOF position in 3D world coordinates of the robot system using translation and rotation of each joint. Due to the intersecting axes there is no translation between joints. Thus, the position of the EOF is

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

Where $\mathbf{q} = (q_1\ q_2\ q_3)^T$ is the joint angle vector, $\mathbf{T}_1^0$ and $\mathbf{T}_{EOF}^3$ are the translations from first coordinate system to world coordinates and EOF to third coordinate system respectively. The rotations are $\mathbf{R}_1^0(q_1)$, $\mathbf{R}_2^1(q_2)$ and $\mathbf{R}_3^2(q_3)$ for first, second and third joint rotation. For our robot this results to

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

where c_1 is $\cos q_1$, s_1 is $\sin q_1$, etc.

Having a sphere as bat, we can ignore the orientation of the end-effector.

3.2 Dynamics

We derive the dynamics for our robot in two steps. First we create the spatial vector notation (Featherstone, 2008; Featherstone, 2010), which we can plug into the Matlab library Spatial_v2 by (Featherstone, 2012). Therefore, we have to provide the connectivity graph, the geometry – which describes the translation and rotation, the joint type (revolute, prismatic, helical) and the spatial inertia for each body.

We approximated the inertia matrices using simple geometry (Figure 4) and masses from a CAD model. Most relevant for the controller is the bat mass of 0.9 kg.

In the second step we obtain the equations of motion in the joint space form

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q},\dot{\mathbf{q}}) + \tau_g(\mathbf{q}) = \tau_m, \quad (3)$$

where $\mathbf{M}(\mathbf{q})$, $\mathbf{c}(\mathbf{q},\dot{\mathbf{q}})$, $\tau_g(\mathbf{q})$ and τ_m are the joint space inertia, Coriolis and centrifugal terms, gravitational terms and motor torques respectively. The notation for the joint angles, velocities and accelerations is $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ (Siciliano and Khatib, 2008).

To obtain the Coriolis and gravitational terms, we use the inverse dynamics of the Spatial_v2 library together with the symbolic toolbox of Matlab and setting the acceleration $\ddot{\mathbf{q}}$ to zero. The remaining joint space inertia is provided by the composite-rigid-body algorithm (Featherstone, 2010).

3.3 Elastic Joints

Our robot has relevant elasticities in the joints which are mainly due to the tooth belt gear. Therefore, we have to enhance the equation of motion (3) by

the elastic joint model, where motor angle, velocity and acceleration are described by $\boldsymbol{\theta}$, $\dot{\boldsymbol{\theta}}$ and $\ddot{\boldsymbol{\theta}}$ respectively (Siciliano and Khatib, 2008; Spong, 1995; Albu-Schaeffer et al., 2007).

The coupling between motor and joint is the tooth belt, which can be approximated by a spring with stiffness **K** and a damping **D**. In the following we set the damping **D** to zero, which is the worst case.

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q},\dot{\mathbf{q}}) + \tau_g(\mathbf{q}) + \mathbf{K}(\mathbf{q}-\boldsymbol{\theta}) = 0 \quad (4)$$

$$\mathbf{B}\ddot{\boldsymbol{\theta}} + \mathbf{K}(\boldsymbol{\theta}-\mathbf{q}) = \tau_m \quad (5)$$

Where **B** and τ_m are motor inertia and torque transformed to the joint side with the gear ratio $n_G = \frac{689}{36}$ by

$$\mathbf{B} = n_G^2 \mathbf{B}_{motor}, \quad (6)$$

$$\tau_m = n_G \tau_{motor}. \quad (7)$$

We stack the dynamics into a nonlinear state space representation $\dot{\mathbf{x}} = f_{\text{dyn}}(\mathbf{x},\mathbf{u})$, defining the state vector $\mathbf{x}$, the input vector $\mathbf{u}$ and the function $f_{\text{dyn}}(\mathbf{x},\mathbf{u})$ as

$$\mathbf{x} = \left(\mathbf{q}^T\ \dot{\mathbf{q}}^T\ \boldsymbol{\theta}^T\ \dot{\boldsymbol{\theta}}^T\right)^T, \quad (8)$$

$$\mathbf{u} = \tau_m, \quad (9)$$

$$f_{\text{dyn}}(\mathbf{x},\mathbf{u}) = \begin{pmatrix} \dot{\mathbf{q}} \\ -\mathbf{M}(\mathbf{q})^{-1}\left(\mathbf{c}(\mathbf{q},\dot{\mathbf{q}}) + \tau_g(\mathbf{q}) + \mathbf{K}(\mathbf{q}-\boldsymbol{\theta})\right) \\ \dot{\boldsymbol{\theta}} \\ -\mathbf{B}^{-1}\mathbf{K}(\boldsymbol{\theta}-\mathbf{q}) + \mathbf{B}^{-1}\tau_m \end{pmatrix}, \quad (10)$$

which we use for our optimal control problem.

Not modeled. At the moment we ignore constraints of joint angles and motor torques, as well as friction.

4 TASK LEVEL OPTIMAL CONTROL

Having the robot model, kinematics of the EOF and the equations of motion derived in the last section, we now want to formulate our task of being at a desired position with given speed in specified time, as a finite horizon optimal control problem. The principle will be explained at first. Furthermore, we explain the control parameters.

4.1 Finite Horizon Controller

The principle of a finite horizon controller is to minimise a cost function in the time interval $[0,T]$, by finding an optimal policy, i. e. mapping from $(\mathbf{x},t) \mapsto \mathbf{u}$. The cost function consists of costs during the process time, depending on state and command, and a final cost, where only the state contributes to the cost. This is due to the finite time horizon, as the optimal command in the last step is $\mathbf{u} = 0$ and any other command would increase the cost. The general cost function is:

$$\text{cost}(\mathbf{x},\mathbf{u}) = \text{cost}_\text{f}(\mathbf{x}(T)) + \int_0^T \text{cost}(\mathbf{x}(t),\mathbf{u}(t))dt. \quad (11)$$

For our optimal control problem we define the final cost in dependency of the position **p** and the velocity **v**. The process costs are then dependent on the input torque and additionally on elastic vibrations which are penalised.

$$\text{cost}(\mathbf{x},\mathbf{u}) = \text{cost}_\text{p}(\mathbf{x}(T)) + \text{cost}_\text{v}(\mathbf{x}(T)) + \int_0^T \left[\text{cost}_\text{vib}(\mathbf{x}(t)) + \text{cost}_\tau(\mathbf{u}(t))\right] dt \quad (12)$$

In the following the cost functions will be derived and explained independently.

4.2 Cartesian Position

The position cost function formalises "be at position **p** at time T". The Cartesian position of the EOF is calculated by Equation (2). We define a new function to penalise the quadratic deviation of actual system position to a desired goal point **p** in world coordinates. Hence, the position penalty is

$$\text{cost}_\text{p}(\mathbf{x}) = q_\text{p}\left(f_\text{kin}(\mathbf{q}) - \mathbf{p}\right)^2, \quad (13)$$

where q_p is the penalisation parameter. The desired point can have any value in 3D space, if it is not reachable by the EOF, the distance to it will be minimised.

4.3 Cartesian Velocity

The velocity cost function formalises "have velocity **v** at time T". Considering our goal to reach a desired position with a given velocity, we can apply a similar penalisation as for the position to the velocity. Thus, we need the derivative of the kinematics with respect to time

$$f_\text{vel}(\mathbf{q},\dot{\mathbf{q}}) = \frac{\partial f_{kin}(\mathbf{q})}{\partial t} = \frac{\partial f_{kin}(\mathbf{q})}{\partial \mathbf{q}}\dot{\mathbf{q}}. \quad (14)$$

We construct the velocity penalty by penalising the quadratic velocity difference from the desired velocity **v** in Cartesian coordinates:

$$\text{cost}_\text{v}(\mathbf{x}) = q_\text{v}\left(f_\text{vel}(\mathbf{q},\dot{\mathbf{q}}) - \mathbf{v}\right)^2. \quad (15)$$

Only velocities in the tangential plane of the sphere's workspace can be achieved, for **v** outside that plane the difference is minimised. Note, that because the tangential plane depends on **p**, this will result in a compromise between **p** and **v** and leading to an increase of cost.

4.4 Vibration Reduction

The vibration reduction cost function formalises "oscillation is unwanted". The flexible joint system tends to oscillate in the spring $(\mathbf{q}-\boldsymbol{\theta})$. We want the controller to avoid this by penalising spring motion. So

$$\text{cost}_{\text{vib}}(\mathbf{x}) = q_{\text{vib}}\left(\dot{\mathbf{q}}-\dot{\boldsymbol{\theta}}\right)^2 \tag{16}$$

is the overall vibration reduction penalisation.

4.5 Motor Torque

The motor torque cost function formalises "minimal effort". We define a cost function that penalises the squared input torque by a factor r_τ. This is a canonical choice of actuation costs, favours torques evenly distributed over time and also corresponds to heat dissipation of the DC-motor which we want to minimise.

$$\text{cost}_\tau(\mathbf{u}) = r_\tau \mathbf{u}^2 \tag{17}$$

5 DYNAMIC PROGRAMMING

The previously defined model and cost functions will now be applied to compute the optimal control policy expressed as a feedback gain matrix $\mathbf{K}_n$. Therefore, we use dynamic programming to compute the optimal control policy by backward induction. It starts with the final cost $\text{cost}_\text{f} = \text{cost}_\text{p}(\mathbf{x}(T)) + \text{cost}_\text{v}(\mathbf{x}(T))$. For each time-step T_s it computes the effective cost of the previous time-step as a function of that state $\mathbf{x}(T-T_s)$ by finding the $\mathbf{x}(T-T_s)$ dependent optimal control $\mathbf{u}(T-T_s)$. This process goes back from $\mathbf{x}(T)$ to $\mathbf{x}(0)$.

The case of dynamic programming we use here is known as the finite horizon linear quadratic regulator (Sontag, 1990). An advantage is the automatic computation of the feedback gain matrix. Moreover, it is a step by step method and can be easily adapted to discrete systems with sample time T_s controlled by an LQR.

The LQR is based on a linear discrete system dynamics

$$\mathbf{x}_{n+1} = \mathbf{A}_n\mathbf{x}_n + \mathbf{B}_n\mathbf{u}_n, \tag{18}$$

$$\mathbf{u}_n = \mathbf{K}_n\mathbf{x}_n, \tag{19}$$

where $\mathbf{A}$ and $\mathbf{B}$ denote the system and input matrix respectively and $n = t/T_s$ is the discrete step time, which is always an integer.

To integrate our system description derived in the last sections into the dynamic programming algorithm using a linear quadratic regulator, we first have to linearise our system and afterwards discretise it to match into (18).

5.1 Linearisation

The equations of motion describe a nonlinear dynamic function $f_{\text{dyn}}(\mathbf{x},\mathbf{u})$. As mentioned the LQR is working only on linear systems, we need to linearise Equation (10). We do this by deriving the first order Taylor series around some linearisation point $\mathbf{x}^*$ and $\mathbf{u}^*$ such that

$$f_{\text{lin}}(\mathbf{x},\mathbf{u}) = f_{\text{dyn}}(\mathbf{x}^*,\mathbf{u}^*) + \frac{\partial f_{\text{dyn}}(\mathbf{x}^*,\mathbf{u}^*)}{\partial \mathbf{x}}(\mathbf{x}-\mathbf{x}^*) + \frac{\partial f_{\text{dyn}}(\mathbf{x}^*,\mathbf{u}^*)}{\partial \mathbf{u}}(\mathbf{u}-\mathbf{u}^*) \tag{20}$$

is the linear system approximation. Determining the linearisation points $\mathbf{x}^*$ and $\mathbf{u}^*$ will be discussed later. As we can see in this Equation, we have a constant term

$$\mathbf{c} = f_{\text{dyn}}(\mathbf{x}^*,\mathbf{u}^*) - \frac{\partial f_{\text{dyn}}(\mathbf{x}^*,\mathbf{u}^*)}{\partial \mathbf{x}}\mathbf{x}^* - \frac{\partial f_{\text{dyn}}(\mathbf{x}^*,\mathbf{u}^*)}{\partial \mathbf{u}}\mathbf{u}^*, \tag{21}$$

which is not handled in (18). To make use of Equation (18), we can combine first and zero order terms into a single matrix. Thus, we get an affine notation (Goretkin et al., 2013), such that the resulting affine notation with

$$\dot{\mathbf{x}} = f_{\text{lin}}(\mathbf{x},\mathbf{u}) = \bar{\mathbf{A}}\bar{\mathbf{x}} + \bar{\mathbf{B}}\mathbf{u}, \tag{22}$$

will later fit into (18) after discretisation, where $\bar{\mathbf{x}} = \begin{pmatrix}\mathbf{x}^T & 1\end{pmatrix}$ and the matrices $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are

$$\bar{\mathbf{A}} = \begin{pmatrix}\mathbf{A} & \mathbf{c}\\ \mathbf{0} & 0\end{pmatrix} = \begin{pmatrix}\frac{\partial f_{\text{dyn}}(\mathbf{x}^*,\mathbf{u}^*)}{\partial \mathbf{x}} & \mathbf{c}\\ \mathbf{0} & 0\end{pmatrix}, \tag{23}$$

$$\bar{\mathbf{B}} = \begin{pmatrix}\mathbf{B}\\ \mathbf{0}\end{pmatrix} = \begin{pmatrix}\frac{\partial f_{\text{dyn}}(\mathbf{x}^*,\mathbf{u}^*)}{\partial \mathbf{u}}\\ \mathbf{0}\end{pmatrix}. \tag{24}$$

We apply the same process of linearisation to obtain the costs. Thus we combine (13) and (15) to $\text{cost}_\text{f} = \text{cost}_\text{p} + \text{cost}_\text{v}$ such that we get the final cost

$$\text{cost}_\text{f}(\mathbf{x}) = \overbrace{\begin{pmatrix}f_{\text{kin}}(\mathbf{x})-\mathbf{p}\\ f_{\text{vel}}(\mathbf{x})-\mathbf{v}\end{pmatrix}}^{f_{\text{pv}}(\mathbf{x},\mathbf{p},\mathbf{v})}{}^T \overbrace{\begin{pmatrix}q_\text{p}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & q_\text{v}\mathbf{I}\end{pmatrix}}^{\mathbf{W}_\text{f}} \begin{pmatrix}f_{\text{kin}}(\mathbf{x})-\mathbf{p}\\ f_{\text{vel}}(\mathbf{x})-\mathbf{v}\end{pmatrix}, \tag{25}$$

where $\mathbf{I}$ denotes the identity matrix.

Then we linearise $f_{\text{pv}}(\mathbf{x},\mathbf{p},\mathbf{v})$ around $\mathbf{x}^*$ and obtain

$$\begin{aligned}\bar{\mathbf{J}}_\text{f} &= \begin{pmatrix}\mathbf{J}_\text{f} & \mathbf{c}_{\text{fpv}}\end{pmatrix}\\ &= \begin{pmatrix}\frac{\partial f_{\text{pv}}(\mathbf{x}^*,\mathbf{p},\mathbf{v})}{\partial \mathbf{x}} & f_{\text{pv}}(\mathbf{x}^*,\mathbf{p},\mathbf{v}) - \frac{\partial f_{\text{pv}}(\mathbf{x}^*,\mathbf{p},\mathbf{v})}{\partial \mathbf{x}}\mathbf{x}^*\end{pmatrix}\end{aligned} \tag{26}$$

and the linearised function

$$f_{\text{pvlin}}(\mathbf{x},\mathbf{p},\mathbf{v}) = \bar{\mathbf{J}}_\text{f}\bar{\mathbf{x}}. \tag{27}$$

To get the final cost in linearised form, we first have to build the square of Matrix $\bar{\mathbf{J}}_\mathrm{f}$ and penalise it by $\mathbf{W}_\mathrm{f}$, resulting to the penalisation matrix

$$\bar{\mathbf{Q}}_\mathrm{f} = \bar{\mathbf{J}}_\mathrm{f}^T \mathbf{W}_\mathrm{f} \bar{\mathbf{J}}_\mathrm{f} = \begin{pmatrix} \mathbf{J}_\mathrm{f}^T \mathbf{W}_\mathrm{f} \mathbf{J}_\mathrm{f} & \mathbf{J}_\mathrm{f}^T \mathbf{W}_\mathrm{f} \mathbf{c}_\mathrm{fpv} \\ \mathbf{c}_\mathrm{fpv}^T \mathbf{W}_\mathrm{f} \mathbf{J}_\mathrm{f} & \mathbf{c}_\mathrm{fpv}^T \mathbf{W}_\mathrm{f} \mathbf{c}_\mathrm{fpv} \end{pmatrix}. \quad (28)$$

Finally, we have to multiply the final penalisation matrix $\bar{\mathbf{Q}}_\mathrm{f}$ by the squared state affine vector $\bar{\mathbf{x}}$, giving us the final linearised cost

$$\mathrm{cost}_\mathrm{flin}(\mathbf{x}) = \bar{\mathbf{x}}^T(T) \bar{\mathbf{Q}}_\mathrm{f} \bar{\mathbf{x}}(T). \quad (29)$$

For the vibration reduction cost we can do the same linearisation as for the final cost. Hence, we have

$$\mathrm{cost}_\mathrm{vib}(\mathbf{x}) = (\overbrace{\dot{\mathbf{q}} - \dot{\hat{\mathbf{\theta}}}}^{f_\mathrm{vib}(\mathbf{x})})^T \overbrace{q_\mathrm{vib} \mathbf{I}}^{\mathbf{W}} (\dot{\mathbf{q}} - \dot{\hat{\mathbf{\theta}}}). \quad (30)$$

Linearising $f_\mathrm{vib}(\mathbf{x})$ around $\mathbf{x}^*$ results in

$$\begin{aligned} \bar{\mathbf{J}} &= \begin{pmatrix} \mathbf{J} & \mathbf{c}_\mathrm{vib} \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial f_\mathrm{vib}(\mathbf{x}^*)}{\partial \mathbf{x}} & f_\mathrm{vib}(\mathbf{x}^*) - \frac{\partial f_\mathrm{vib}(\mathbf{x}^*)}{\partial \mathbf{x}} \mathbf{x}^* \end{pmatrix}. \end{aligned} \quad (31)$$

The vibration penalisation $\bar{\mathbf{Q}}$ in linearised form is then

$$\bar{\mathbf{Q}} = \bar{\mathbf{J}}^T \mathbf{W} \bar{\mathbf{J}} = \begin{pmatrix} \mathbf{J}^T \mathbf{W} \mathbf{J} & \mathbf{J}^T \mathbf{W} \mathbf{c}_\mathrm{vib} \\ \mathbf{c}_\mathrm{vib}^T \mathbf{W} \mathbf{J} & \mathbf{c}_\mathrm{vib}^T \mathbf{W} \mathbf{c}_\mathrm{vib} \end{pmatrix}, \quad (32)$$

which gives us the linearised vibration cost

$$\mathrm{cost}_\mathrm{viblin}(\mathbf{x}) = \bar{\mathbf{x}}^T(t) \bar{\mathbf{Q}} \bar{\mathbf{x}}(t). \quad (33)$$

For the torque cost in linearised form we can write the result directly as a squared linear function of the penalty. Thus, the cost is

$$\mathrm{cost}_\tau = \mathbf{u}^T(t) \overbrace{r_\mathrm{tau} \mathbf{I}}^{\mathbf{R}} \mathbf{u}(t), \quad (34)$$

where $\mathbf{R}$ is the penalisation matrix of the input torque.

5.2 Discretisation

By now we have a linearisation of our continuous system model. We now have to discretise it to fit into (18). The discretisation of a first order differential equation

$$\dot{\mathbf{z}} = \mathbf{F}\mathbf{z} \quad (35)$$

can be solved by using the approach

$$\mathbf{F}_n = \mathrm{e}^{\mathbf{F} T_s}, \quad (36)$$

where T_s is the sample time and $\mathrm{e}^{\mathbf{F}T_s}$ is the matrix exponential (Berg et al., 2011). The result is the discrete system

$$\mathbf{z}_{n+1} = \mathbf{F}_n \mathbf{z}_n. \quad (37)$$

Adapting (37) with $\mathbf{F} = \left(\begin{smallmatrix} \bar{\mathbf{A}} & \bar{\mathbf{B}} \\ \mathbf{0} & \mathbf{0} \end{smallmatrix}\right)$ and $\mathbf{z} = \begin{pmatrix} \bar{\mathbf{x}}^T & \mathbf{u}^T \end{pmatrix}^T$ and moreover using a first order Taylor series approximation for the matrix exponential we get the discrete system

$$\begin{pmatrix} \bar{\mathbf{A}}_n & \bar{\mathbf{B}}_n \\ \mathbf{0} & \mathbf{I} \end{pmatrix} = \mathbf{I} + \mathbf{F} T_s \approx \mathrm{e}^{\mathbf{F} T_s}, \quad (38)$$

in the form of (18).

5.3 Feedback Gain

The computation of the feedback gain $\bar{\mathbf{K}}_n$ can be done recursively backward in time using dynamic programming, as explained in (Sontag, 1990). Therefore, the cost function

$$\mathrm{cost}(\mathbf{x}, \mathbf{u}) = \bar{\mathbf{x}}_N^T \bar{\mathbf{Q}}_\mathrm{f} \bar{\mathbf{x}}_N + \sum_{n=0}^{N-1} \bar{\mathbf{x}}_n^T \bar{\mathbf{Q}} \bar{\mathbf{x}}_n + \mathbf{u}_n^T \mathbf{R} \mathbf{u}_n, \quad (39)$$

is minimised leading to the algebraic discrete Riccati Equation

$$\begin{aligned} \bar{\mathbf{P}}_n = \bar{\mathbf{Q}} + \bar{\mathbf{A}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{A}}_n - \\ \bar{\mathbf{A}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{B}}_n (\mathbf{R} + \bar{\mathbf{B}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{B}}_n)^{-1} \bar{\mathbf{B}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{A}}_n \end{aligned} \quad (40)$$

where $\bar{\mathbf{P}}_N$ is set to $\bar{\mathbf{Q}}_\mathrm{f}$ for the final discrete step $N = T/T_s$. In the same iteration step we get the feedback gain matrix

$$\bar{\mathbf{K}}_n = -(\mathbf{R} + \bar{\mathbf{B}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{B}}_n)^{-1} \bar{\mathbf{B}}_n^T \bar{\mathbf{P}}_{n+1} \bar{\mathbf{A}}_n. \quad (41)$$

This is the core optimisation of the overall optimal control algorithm.

5.4 Nonlinear Iterations

To solve the problem of hitting the ball at a specified position with specified velocity at a specified time we use nonlinear iterations. Thus, in the first iteration we compute an initial guess of our nonlinear system (10) from the actual time-step $n = 0$ till the final time-step N. Based on this prediction the linearisations are done and the feedback gains are provided for all future steps. In the next iteration loop, the current system state $\mathbf{x}(0)$ is used together with the linearisations and feedback gains from the last step to predict the behaviour for time-steps $n = 1$ to N and again linearise the system based on this initial guess and providing the feedback gains. In each iteration the linearised system converges more to the nonlinear system. Moreover, if the linearised system diverges from the nonlinear system, the next iteration update step will tend the linearisation towards the nonlinear system as the nonlinear current state $\mathbf{x}(0)$ is used as start parameter. The cascaded implementation is explained in Algorithm 1. The operating loop runs until the ball has been hit. The most important steps are done in *systemForecast*, *lineariseFpv* and *dplqrAffine*.

In *systemForecast* we compute the linearised discrete affine system $\bar{\mathbf{A}}_n$ and $\bar{\mathbf{B}}_n$ in three steps.

First, we predict the system behaviour and linearisation points $\mathbf{x}_n^*$ and $\mathbf{u}_n^*$ with (18) and (19) given $\bar{\mathbf{K}}_n$, $\bar{\mathbf{A}}_n$ and $\bar{\mathbf{B}}_n$ from the last iteration step. The linearisation of the system is also done in *systemForecast*. Hence, we create the first order Taylor series in (21),

Algorithm 1: Cascaded implementation.

```
   // Eq. (42)-(45)
1  [Ā_n, B̄_n, Q̄_f, x*_n] =initialGuess(p, v, x(0));
2  for n_up = [0, N−1] do
3  |  x(0) =getState();
4  |  if n_up > 0 then
   |  |  // Eq. (18), (19), (21), (23), (24), (38)
5  |  |  [Ā_n, B̄_n, x*_n] =systemForecast(K̄_n, x(0), Ā_n, B̄_n);
6  |  |  Q̄_f =lineariseFpv(x*_N, p, v); // Eq. (26), (28)
7  |  end
   |  // Eq. (40), (41)
8  |  K̄_n =dplqrAffine(Ā_n, B̄_n, Q̄_f, Q̄, R);
9  |  setGain(K_n);
10 end
```

(23) and (24) using the provided linearisation points to get the continuous system at those points. Finally, we discretise the continuous system by (38).

Secondly, in the function *linearisFpv* we obtain the final penalisation with $\mathbf{x}_N^*$ as linearisation point of Equation (26). The resulting matrix $\bar{\mathbf{J}}_f$ is then penalised by $\mathbf{W}_f$ leading to the final penalisation matrix $\bar{\mathbf{Q}}_f$ (Equation (28)).

Finally, the feedback gain for the controller is calculated in *dplqrAffine* based on the linearisations via (40) and (41).

5.5 Initial Guess

Before iterating the loop of Algorithm 1 we need to provide start conditions. This is fulfilled in *initialGuess*. As we have no actual control gains or system linearisations at the iteration start, we need to provide a first system prediction. Thus, we first calculate an initial guess for a desired final state $\mathbf{x}_d$ from $\mathbf{p}$ and $\mathbf{v}$ which is then used to guess the intermediate states. This is done by iterating the desired position and apply first axis rotation $\mathbf{R}_0^1(q_1)$ for different angles q_1 in the range $[-170°, 170°]$ utilising 5° steps. Obtaining

$$\mathbf{p}^2 = \mathbf{T}_0^2 + \mathbf{R}_0^1(q_1)\mathbf{p}^0 \tag{42}$$

as desired point in Axis 2 coordinates, except any rotations of Coordinate System 2. Next we solve the inverse kinematics for given q_1 by

$$q_{\text{invkin}}(q_1, \mathbf{p}) = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} q_1 \\ \arctan\left(\frac{p_x}{p_z}\right) \\ \arcsin\left(\frac{p_y}{\sqrt{p_x^2+p_y^2+p_z^2}}\right) \end{pmatrix}. \tag{43}$$

This results in a set of possible angle rotations reaching the same end-position $\mathbf{p}$. Out of the set we want to find the one with the lowest overall rotation

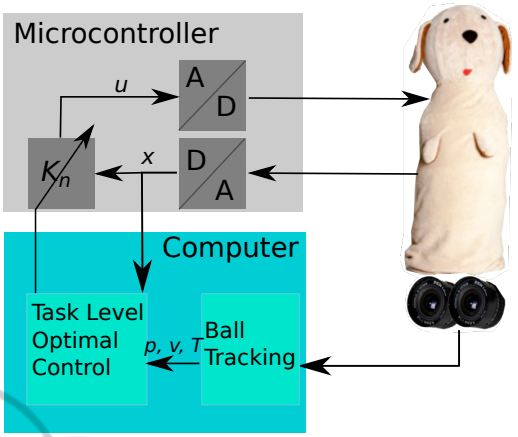

Figure 5: Distributed system with robot as plant, a microcontroller as feedback controller and collector of sensor signals from the robot and a computer as ball detector, using the cameras, and feedback gain optimiser.

of the joints, i. e.

$$\mathbf{q}_d = \min_{q_1} \sum (\mathbf{q}(0) - q_{\text{invkin}}(q_1, \mathbf{p}))^2, \tag{44}$$

where $\mathbf{q}(0)$ is the current joint position.

For the angular velocity we plug $\mathbf{q}_d$ into the Jacobian of (14). Taking the pseudo-inverse, as the inverse does not exist due to redundancy, results in the desired velocity on the tangential plane

$$\dot{\mathbf{q}}_d = \left(\frac{\partial f_{kin}(\mathbf{q}_d)}{\partial \mathbf{q}}\right)^{\dagger} \mathbf{v}_d, \tag{45}$$

where $()^{\dagger}$ denotes the pseudo-inverse.

To provide the discrete linearised system we assume a straight line from current state $\mathbf{x}(0)$ to the initial desired state $\mathbf{x}_d$ as linearisation points $\mathbf{x}_n^*$. Though, we suppose that the system will follow the states on the line without any commanded torque ($\mathbf{u}_n^* = 0$). With these definitions we can compute the linearised system by (21), (23) and (24) and descretise the result by (38). Note, that this is just an initial guess which has no influence to the final optimal result as long as it takes Algorithm 1 to the valley at the global optimum.

5.6 Chained Implementation

Computing a feedback gain instead of a trajectory allows us to modify the system presented in the first section in Figure 3 to a faster distributed system in Figure 5. Our described computations run separated on a computer and microcontroller using discrete signals from camera and other sensors.

The microcontroller is responsible for the joints torque control. It computes the torque using the feedback gain matrix $\bar{\mathbf{K}}$ and the actual system state $\mathbf{x}(0)$, running on a sample time of 1 ms.

Figure 5 also shows the camera system connected to the computer which tracks throwing balls, which is not part of this paper but basically explained in Section 7. Moreover, velocity, position and time of impact are predicted and provided and serve as our controlling system goal parameters. The camera system runs on a slower sample time of 50 ms. The slower camera sample time is used as update time T_{up} in the loop of Algorithm 1. During each camera update one iteration is processed. This allows us a forecast of the states $\mathbf{x}_n^*$, given the input $\mathbf{u}_n^*$. The forecasted states are then used as new linearisation points for the system at the next camera update and a more precise calculation of the controllers gain matrices $\bar{\mathbf{K}}_n$ can be achieved. The feedback gains are transferred after each iteration to the microcontroller. This allows a faster sampling time, as the microcontroller just computes one matrix multiplication in each step. If it had to compute the feedback gain matrix, such a fast sample time would be impossible. Moreover, it allows a fast reaction on disturbances in a linearised optimal way and later converges to the nonlinear optimum by the iteration loop.

6 COMPARISON TO INFINITE HORIZON LQR

Compared to the infinite horizon LQR, which has no final time, we make use of the finite LQR by setting the final time as input parameter to the optimisation problem. Moreover, the infinite LQR would lead to a fast achieving of the final state, which causes greater amount of input command. This is not useful with the time horizon in mind, where we know when to reach the final state and so we are able to distribute the commanded torques more equally to each step. We will show this in the next section by running a simulation of our system and the distribution of a fast running controller on a microcontroller and a slower running feedback gain computation on a computer.

7 SYSTEM CONTEXT

In this paper we present the controller itself using the position $\mathbf{p}$, velocity $\mathbf{v}$ and the final time T as input parameters for the controller. How to obtain these values is not part of this paper. However, for a better understanding of the overall process we give a short description about the system context. Throwing balls towards the robot will lead to a detection of the balls using two cameras mounted left and right from the robots center axis. Circles are detected independently in the left and right camera image. The detected circles are then passed to either a Multiple Hypothesis Tracker (MHT) that handles clutter and covers missing detections followed by an Unscented Kalman Filter (UKF) for estimation of position and velocity, which are used for prediction of the future states (Laue et al., 2013). Alternatively they are passed to a so called Fully Probabilistic Multiple Target Tracker (FPMHT)(Birbach and Frese, 2013). With both algorithms we intersect the predicted trajectory with the workspace sphere and use the first intersection as $\mathbf{p}$. This is implemented on the previous version of the robot, with $\mathbf{v}$ fixed to 0. The velocity needed for return is future work, but the theoretical implementation has been investigated in (Hammer, 2011). Therefore, a realistic rebound model of two balls can be used to determine the direction vector and so the velocity needed to rebound the ball.

8 SIMULATIONS

In a few scenarios we demonstrate the system behaviour using TLOC to reach a specified position and velocity on different time constraints and we illustrate the robustness by applying a disturbance torque to the system. All simulations are based on the dataflow diagram of Figure 5 implemented in Matlab and Simulink. There are three simulations combined: A robot simulation, which runs with the forward dynamics of Spatial_v2 library and some extensions for the flexible case; a microcontroller simulator, including only the torque computation in each millisecond and discretisation via sample and hold; the computer simulator, for operating the TLOC every 50 ms.

8.1 Elasticity Behaviour

First we illustrate the controller's behaviour and system reaction by reaching a defined goal position $\mathbf{p}_{\text{d}}$, with velocity $\mathbf{v}_{\text{d}}$ at time T. Therefore, we set the penalisation for position q_{p}, velocity q_{v}, torque r_τ and vibration reduction q_{vib}. The values for the parameters are shown in Table 1. Starting in upright position $\mathbf{p}(0)$ with zero velocity $\mathbf{v}(0)$ we can see the behaviour of the joints and the commanded torque in Figure 6. We see the motor and joint positions and velocities as well as the commanded torque. Due to the spring stiffness K and the coupling between motor and joint $(\mathbf{q} - \boldsymbol{\theta})$, we can see that the commanded torque acts fast on the motor velocity, but the reaction of the joints velocity is delayed. This delayed behaviour explains the peaks in the commanded torque. At the first linearisation points the system does not react as the con-

Table 1: Parameters for the elasticity behaviour.

Parameter	Value	
$\mathbf{p}(0)$	$(0.0\ 0.0\ 1.9)^T$	m
$\mathbf{p}_\mathrm{d}$	$(0.45\ 0.636\ 1.45)^T$	m
$\mathbf{v}(0)$	$(0.0\ 0.0\ 0.0)^T$	m/s
$\mathbf{v}_\mathrm{d}$	$(0.0\ -1.0\ 1.414)^T$	m/s
T	0.5	s
q_p	5×10^6	m^{-2}
q_v	5×10^6	$\mathrm{s}^2\mathrm{m}^{-2}$
q_vib	1.0	s^2
r_τ	0.01	$(\mathrm{Nm})^{-2}$

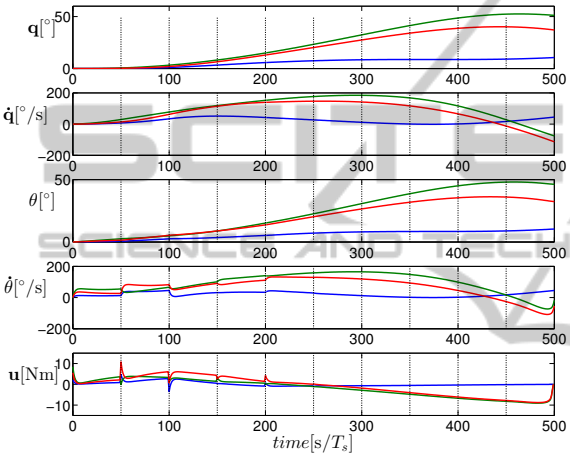

Figure 6: Reaching the goal position in $T = 0.5\,\mathrm{s}$. Blue, green, red denotes Axis 1, 2 and 3 respectively. Vertical dashed lines denote the update times.

troller wanted due to linearisation errors, thus, to fulfil a reaction on the joints position and velocity, the spring has to be tensioned or loosened quickly. After reaching the desired tension on the spring the torque can be smoother.

Moreover, the torque is distributed over the whole time span, bringing the axes to a position where the goal could be reached. Here we accurately reach the final position $\mathbf{p}(T) = (0.45\ 0.6359\ 1.4507)^T$ m and velocity $\mathbf{v}(T) = (-0.0007\ -1.0004\ 1.4121)^T$ m/s with the angular positions and velocities of Figure 6.

8.2 Time and Disturbance Behaviour

In this section we show the behaviour of the system when the time horizon changes and if a torque is added to the system as disturbance. Thus, we hit the ball in every scenario in the same desired position $\mathbf{p}_\mathrm{d}$, with velocity $\mathbf{v}_\mathrm{d}$ and desired time T. Starting in the upright EOF position $\mathbf{p}(0)$ with zero velocity $\mathbf{v}(0)$. We set the penalisation for position q_p, velocity q_v, torque r_τ and vibration reduction q_vib and other values as shown in Table 2.

Table 2: Parameters for the time and disturbance behaviour.

Parameter	Value	Unit
$\mathbf{p}(0)$	$(0.0\ 0.0\ 1.9)^T$	m
$\mathbf{p}_\mathrm{d}$	$(0.0\ 0.0\ 1.9)^T$	m
$\mathbf{v}(0)$	$(0.0\ 0.0\ 0.0)^T$	m/s
$\mathbf{v}_\mathrm{d}$	$(5.0\ 0.0\ 0.0)^T$	m/s
T	0.3	s
q_p	5.0×10^6	m^{-2}
q_v	5.0×10^6	$\mathrm{s}^2\mathrm{m}^{-2}$
q_vib	1.0	s^2
r_τ	0.01	$(\mathrm{Nm})^{-2}$

Table 3: Comparison of the experiments of reached position and velocity to desired reference. Denoted in the table as ref.

	Position [mm]	Velocity [m/s]
ref	$(0.0, 0.0, 1900)$	$(5.0, 0.0, 0.0)$
(a)	$(1.2, -0.1, 1900)$	$(4.9989, -0.0001, -0.0067)$
(b)	$(0.5, -0.1, 1900)$	$(4.9996, -0.0001, -0.0029)$
(c)	$(1.1, -0.9, 1900)$	$(4.9992, -0.0018, -0.0063)$

The position and velocity results of our experiments (a)–(c) are given in Table 3 . All figures including the joint angle, joint velocity and the input command. Here we neglect the motor position and velocity in the figures as well as the vertical lines showing the update. The update time has not changed and is again 50 ms.

(a) We first want to reach the desired position and velocity in the given time of 0.3 s as comparison example. Therefore, Axis 2 turns backwards to accelerate the axis to the desired velocity (c. f. Figure 7). Position and velocity are accurately reached.

(b) To show the time benefit of the controller, we change the desired time to $T = 0.8\,\mathrm{s}$. This results in a reduction in torque and "striking out" more from the EOF rest position compared to the previous simulation (a). The further Axis 2 moves away from the rest position the better it can accelerate the EOF by less command torque. This is the result of giving the controller more time to reach the goal, thus it can distribute the torque better over time (Figure 8).

(c) We now show the robustness of our controller by applying a disturbance torque such that the input torque is $\tau_m = \mathbf{u} + \tau_\mathrm{dist}$. This can also be interpreted

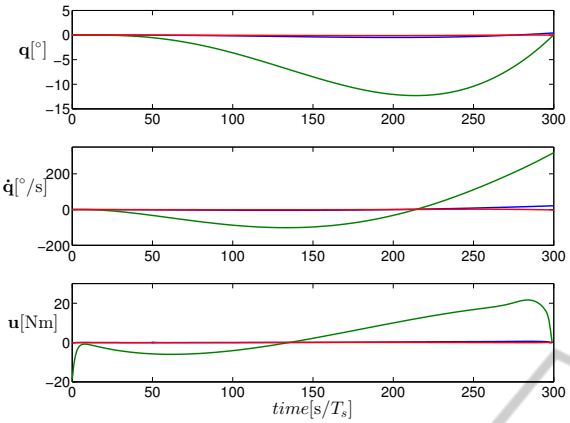

Figure 7: Simulation (a), reaching the goal position and velocity in $T = 0.3$ s. Blue, green, red denotes Axis 1, 2 and 3 respectively.

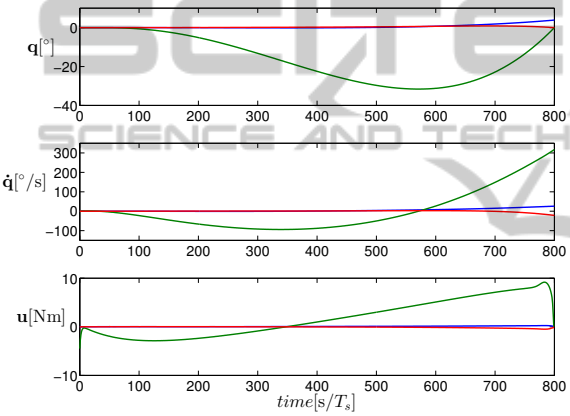

Figure 8: Simulation (b), reaching the goal position and velocity in $T = 0.8$ s. Blue, green, red denotes Axis 1, 2 and 3 respectively.

as a torque resulting from friction, which is demonstrated later. A disturbance torque can be, e. g. some external force pushing the head to another direction. Figure 9 shows the disturbance as a step function of $\tau_{\text{dist}} = (10\ 0\ 2)^T$ Nm starting after 54 ms. Here you can see the difference of forecast behaviour and real behaviour by little spikes in the commanded torque at the update steps. The reason is that the linearisation points $\mathbf{x}_n^*$ are based on the modeled behaviour. When the nonlinear system deviates from that behaviour, the controller immediately reacts in a linearised optimal way, because the output of dynamic programming is not a sequence of torques but a policy, i. e. a sequence of controller gains. In addition to this immediate linearised response, in the next iteration of Algorithm 1 the measured state $\mathbf{x}(0)$ is used, linearisations are updated and the policy is recomputed based on that state. This results in a (close to) nonlinear optimal response from that point on, visible in the small spike in $\mathbf{u}$.

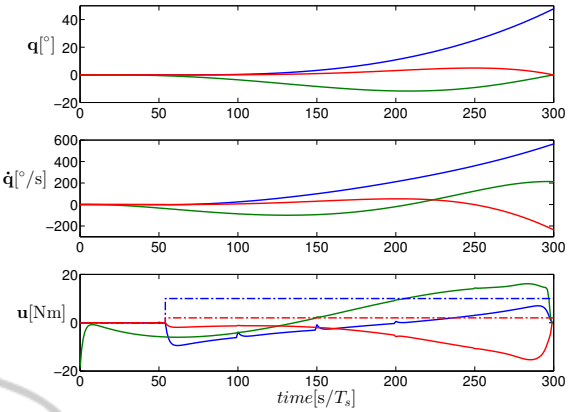

Figure 9: Simulation (c). Reaching the goal position and velocity in $T = 0.3$ s and applying a disturbance torque to Axis 1 (dashed blue) and 3 (dashed red). Blue, green, red denotes Axis 1, 2 and 3 respectively.

However, the goal position is met accurately (c. f. Table 3) even if a disturbance is acting on the system. Unlike trajectory optimisation followed by position control the system, however, does not hold its trajectory (compare Figure 7 and 9) but flexibly reacts to the disturbances even distributing the motion in a different way. The controller adapts to the new situation. Furthermore, the desired position and velocity are hold accurately, which would not be the case if a joint torque trajectory would have been set to the system.

8.3 Friction

In this simulation we want to show how the controller deals with friction, which is unconsidered in the model but added to the plant. The goal position and velocity is given in Table 3. We set the static friction to 5 Nm and the Coulomb friction to 4 Nm. As friction is a non-differentiable function, which is a problem for the simulation solver, we approximate the friction by a sigmoid function to get a soft transition when the velocity is around zero. In this experiment we let the friction act on the motor side, such that a zero crossing of the input torque $\mathbf{u}$ leads to a change in the motor acceleration $\ddot{\boldsymbol{\theta}}$. If the resulting velocity changes its direction, it can be seen in the friction too. If the velocity is around zero, then friction is between minimum and maximum static friction (c. f. Figure 10 the red and blue dashed lines). Compared to the real world we differ in the correctness of the friction model, but we are interested in moving, so we do not need to be accurate when the friction is highest, which is at zero velocity. Moreover, we reached the position $(1.0, 0.0, 1900)$ [mm] and velocity $(4.9982, 0.0006, -0.0056)$ [m/s] accurately, which

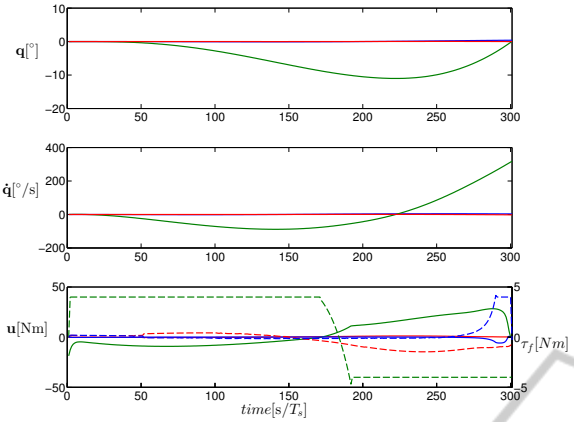

Figure 10: Reaching the goal position and velocity in $T = 0.3\,\mathrm{s}$ and applying friction to all axes (dashed). Blue, green, red denotes Axis 1, 2 and 3 respectively.

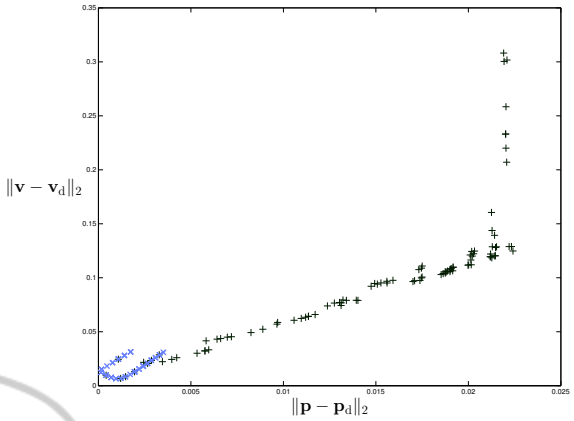

Figure 11: Euclidean distance from reached position and velocity to desired one with varied parameters **B** and **K** within $\pm 10\%$(x markers) and additionally with **D** (+ markers).

shows the robustness also against friction which is not considered in the model.

8.4 Deviation of Model Parameters

In the previous tests we expected that the model of the plant and the plant itself are identically. In this simulation we identify how the controller is acting, if the plant parameters are varied, i. e. varying the values for motor inertia **B**, joint spring stiffness **K** and damping **D**. First, we run the simulations 100 times and change the parameters **B** and **K** in the range of $\pm 10\%$ of the modeled value. Secondly, we repeat the test and additionally vary the damping **D** coefficient within $[0, \mathbf{K} \cdot 0.1\mathrm{s}]$.

Figure 11 points out, that a divergence of the parameters will lead to a deviation of the accuracy. The damping parameter seems more critical (+ markers), but the error still be below 2.5 cm, which is precise enough to hit the ball despite considerabel velocity error.

8.5 Example of Intended Use

Finally, we provide a video[1] where four motion sequences are shown. Each sequence consists of a movement reaching a desired position with given velocity at a specified time and moving back to the rest upright position with velocity zero. Then the next sequence starts. This simulates the ball batting task.

In such a task the robot waits for thrown balls in its initial position, then accelerates towards the balls intersection point. Hitting the ball with desired velocity should return the ball to the player thrown the ball or towards another person standing next to the robot.

The video just simulates the robot motion and position. The specified position is a ball and the desired velocity is marked as an arrow. The arrow direction shows the desired Cartesian velocity direction. If the desired velocity is set to zero, no arrow will be drawn.

It is shown that the movements include all axes. Moreover, the dynamic of the robot can be seen. It shows fast and slow movements, depending on the desired time for a movement and the specified velocity.

9 CONCLUSION AND FUTURE WORK

We have shown an optimal state feedback controller for a simulated, redundant and flexible robot. The controller is capable of steering the joints position and velocity such that they are reaching a desired Cartesian position under and velocity at a given time, which is needed to rebound a ball. Moreover, the controller intelligently distributes the torques to all axes by making use of the redundancy and hence, having less maximum torque for a single axis. Additionally, we see an adaption of the controller to changing conditions.

The presented controller will be extended by constraints on joint angles and motor torques - which were neglected in this paper -, to adapt our approach more to the physical system. Furthermore, we have to take friction into consideration, which is challenging in slow joint movements, but we have shown that the controller can handle it without knowing the friction. Finally, the whole process has to be implemented on the real robot, including system identifi-

[1] http://www.informatik.uni-bremen.de/agebv2/downloads/videos/schuethe_icinco_14_doggyExample.mp4

cation and state estimation, to verify our simulated results. For playing back balls accurately we have to calibrate the system to get a good fitting model. A very worthwhile further extension could be to include the physics of playing back the ball into the optimisation, so the system does not try to achieve a position and velocity as explained herein, it would optimise the ball's goal position, e. g. an opponent player's position or any other desired position surrounding the robot.

ACKNOWLEDGEMENTS

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

Albu-Schaeffer, A., Ott, C., and Hirzinger, G. (2007). A unified passivity-based control framework for position, torque and impedance control of flexible joint robots. *The International Journal of Robotics Research*, 26(1):23–39.

Berg, J., Patil, S., Alterovitz, R., Abbeel, P., and Goldberg, K. (2011). LQG-based planning, sensing, and control of steerable needles. In Hsu, D., Isler, V., Latombe, J.-C., and Lin, M., editors, *Algorithmic Foundations of Robotics IX*, volume 68 of *Springer Tracts in Advanced Robotics*, pages 373–389. Springer Berlin Heidelberg.

Birbach, O. and Frese, U. (2013). 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*, pages 4746–4751.

Featherstone, R. (2008). *Rigid body dynamics algorithms*, volume 49. Springer Berlin.

Featherstone, R. (2010). A beginner's guide to 6-D vectors (part 2) [tutorial]. *Robotics Automation Magazine, IEEE*, 17(4):88–99.

Featherstone, R. (2012). Spatial_v2 (version 2). http://royfeatherstone.org/spatial/v2/notice.html.

Goretkin, G., Perez, A., Platt, R., and Konidaris, G. (2013). Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2429–2436.

Hammer, T. (2011). Aufbau, Ansteuerung und Simulation eines interaktiven Ballspielroboters. Master's thesis, Universitaet Bremen.

Hu, J.-S., Chien, M.-C., Chang, Y.-J., Su, S.-H., and Kai, C.-Y. (2010). A ball-throwing robot with visual feedback. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2511–2512.

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

Laue, T., Birbach, O., Hammer, T., and Frese, U. (2013). An entertainment robot for playing interactive ball games. In *RoboCup 2013: Robot Soccer World Cup XVII*, Lecture Notes in Artificial Intelligence. Springer. to appear.

Muelling, K., Kober, J., Kroemer, O., and Peters, J. (2013). Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279.

Nakai, H., Taniguchi, Y., Uenohara, M., Yoshimi, T., Ogawa, H., Ozaki, F., Oaki, J., Sato, H., Asari, Y., Maeda, K., Banba, H., Okada, T., Tatsuno, K., Tanaka, E., Yamaguchi, O., and Tachimori, M. (1998). A volleyball playing robot. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1083–1089 vol.2.

Perez, A., Platt, R., Konidaris, G., Kaelbling, L., and Lozano-Perez, T. (2012). LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2537–2542.

Reist, P. and Tedrake, R. (2010). Simulation-based lqr-trees with input and state constraints. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5504–5510.

Senoo, T., Namiki, A., and Ishikawa, M. (2006). Ball control in high-speed batting motion using hybrid trajectory generator. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1762–1767.

Siciliano, B. and Khatib, O. (2008). *Springer handbook of robotics*. Springer.

Sontag, E. D. (1990). *Mathematical control theory: deterministic finite dimensional systems*. Texts in applied mathematics ; 6. Springer, New York [u.a.].

Spong, M. W. (1995). Adaptive control of flexible joint manipulators: Comments on two papers. *Automatica*, 31(4):585 – 590.

Zhang, P.-Y. and L, T.-S. (2007). Real-time motion planning for a volleyball robot task based on a multi-agent technique. *Journal of Intelligent and Robotic Systems*, 49(4):355–366.