Pendulum Acrobatics
(Integrated Project 2)

Florian Wolf[1], Pascal Klink[2] and Kai Ploeger[2]

Abstract— **For decades, cart-pole systems have been an important benchmark problem for dynamics and control theory. Due to their high level of nonlinearity, instability and underactuation, they require fast-reactive controllers and are widely used to test emerging control approaches. However, the possible motion profiles of the classical cart-pole system are limited by the rectilinear cart motion. Furthermore, the system is typically investigated in isolation, directly allowing the cart to be displaced. In this work, we investigate a three-dimensional spherical cart-pole system that is realized at the end effector of a Barrett WAM robotic arm, allowing for more challenging motions to be generated while simultaneously introducing a kinematic subtask in the control problem. As we probe the dynamics of this extended system, our experiments show that the simpler gain tuning of the classical LQR controller is preferable over the more intricate MPC approach. The absence of time delays eliminates the need for complex state estimation techniques to compensate for time losses due to parallel execution. Moreover, the LQR controller's ability to promptly respond to state disturbances proves advantageous and allows to convincingly perform stabilization and trajectory tracking tasks. The system operates comfortably at a control frequency of $120\,\mathrm{Hz}$ providing a high temporal resolution for control actions. Supplementary material is available under [here].**

I. Introduction

Since its first mention by Widrow and Smith [1] in 1964, the cart-pole system and its numerous variations since then serve as a benchmark system for all kinds of classical control algorithms and learning methods. Despite being easy to describe, simple to understand and almost all humans can balance a stick on the fingertip or palm of the hand, cart-pole systems stand out by requiring quick control responses due to its inherent nature of nonlinearity, instability and underactuation.

The remainder of this paper is structured as follows: After a short literature review, we propose a mathematical formulation of the tracking control problem, shortly summarize our simulation results of the previous Integrated Project, present various steps in order to close the Sim2Real gap and ultimately show simulation and experimental results.

A. Related work

Prior examples of performing tracking control and stabilization control in simulation include [2, 3] for which we provide an in-depth overview in [4]. In summary, the controllers presented in simulation exclusively worked on spatial inverted pendulums modeled with in integral slider and thus lack the three-dimensional component to move along the z-axis.

Besides results in simulation, Chung et al. [5] in 1999 and Albouy and Praly in 2000 [6] present controllers for stabilization respectively swing-up control on a real system. The former with a three-dimensional spherical pendulum on a simplified arm-like robot and the latter is implemented on a kinematically redundant planar-direct-drive robot in the xy-hyperplane. In both papers they used Linear-quadratic regulator (LQR) controllers and neglect friction effects.

In 2001, Schreiber et al. [7] propose a stabilization controller for a redundant robot based on Nullspace Motions by using absolute joint encoders to measure the position and velocity of the inverted pendulum.

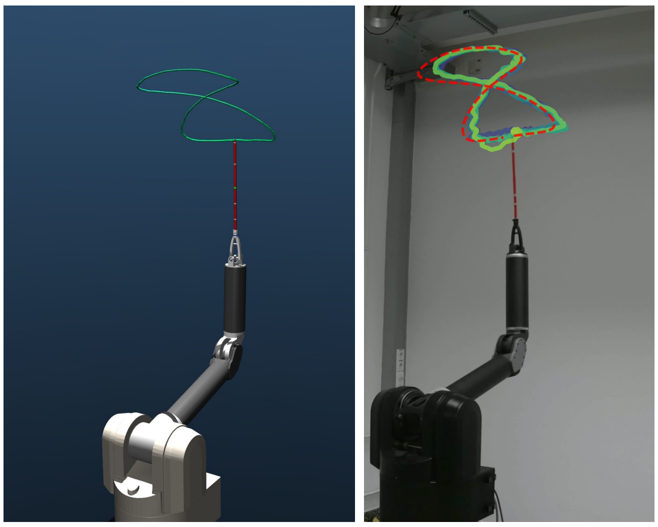


Fig. 1: Trajectory following task with the Barrett WAM in simulation (left) and the real system in the IAS laboratory (right).

In contrast, Marco et al. use an external motion capture system to measure the pendulum's position and velocity, in order to stabilize a simplified one-dimensional inverted pendulum on a humanoid upper-body robot using Gaussian Processes for LQR controllers [8] and Model based policy search for tuning PID controllers [9].

In a more recent work, Vu et al. in 2021 [10] present a LQR-based control technique by using a database of near-time optimal precomputed trajectories to swing-up an inverted pendulum on a 7 degrees of freedom (DoFs) industrial robot. Similar to Schreiber et al., they use absolute joint

*Summer Semester 2023, Compiled on August 29, 2023

[1]Department of Mathematics, Technische Universität Darmstadt, Darmstadt, Germany florian.wolf@stud.tu-darmstadt.de

[2]Intelligent Autonomous Systems, Technische Universität Darmstadt, Darmstadt, Germany

encoders to track the pendulum's position and neglect friction effects. While the offline phase of precomputing optimal trajectories allows an impressive average online computation time of 200 ms by using a K-D tree search, the controller is by design limited to time-stationary target states.

Despite working on a different task than we do, it is noteworthy to analyze the paper by Heins and Schoellig in 2023 [11] which proposes a solution to the waiter's problem. Similar to our approach, the presented method is based on a MPC controller with internal L^2-penalty functions and a Sequential Quadratic Programming (SQP) solver, providing policy updates with a computation time of 10 ms up to 25 ms. Likewise, the MPC controller also allows fast tipping movements to follow optimal trajectories and dynamically avoid obstacles. To obtain position feedback for the base plate a motion capture system running at 250 Hz combined with a Kalman filter is used. Contrastingly, given the difference in the task, Heins and Schoellig's work involves a mobile robot, which distinguishes it from our approach. Futhermore, the biggest difference is the inherently stable state of the presented framework.

B. Mathematical Problem Formulation

The aforementioned problem of tracking control can be formulated as follows:

> Follow a reference trajectory in 3D space with the pendulum's tip while keeping the 3D spherical inverted pendulum in a reasonably balanced state.

Trivially, this formulation also encodes the stabilization task as a simplified version, by keeping the reference trajectory constant.

In all settings, the variables x^{tip} and x^{endeff} describe the pendulum's tip position respectively the base pole's (end effector) center in global coordinates, i.e. the world frame, and we use the variable θ for the angle of the pendulum with the z-axis of the world frame to measure whether the pendulum is balanced.

The following definition reformulates the problem in a more formal way.

Definition 1 (Aim of the project): Assume we are given a time horizon $T \in \mathbb{R}_{>0}$ and a sequence of reference points $(\mathbf{p}_t)_{t\in[0,T]} \subset \mathbb{R}^3$ in the task space. First, we use Crocoddyl's Feasibility Driven DDP (FDDP) solver, to precompute an optimal solution trajectory, consisting of pendulum angles $(\theta_t^{\mathsf{PP}})_{t\in[0,T]}$ and task space positions $(\mathbf{x}_t^{\mathsf{PP},\text{tip}})_{t\in[0,T]}$.

Next, our goal is to generate joint configurations $(\mathbf{q}_t)_{t\in[0,T]}$ and torques $(\mathbf{u}_t)_{t\in[0,T]}$ satisfying pointwise box constraints given by a lower and upper limit $\mathbf{u}_{\min} \leq \mathbf{u}_{\max}$ such that the corresponding pedulum's tip positions $(\mathbf{x}_t^{\text{tip}})_{t\in[0,T]}$ are close to the precomputed reference positions and the pendulum's angles $(\theta_t)_{t\in[0,T]}$ with the z-axis are close to zero.

Based on our findings in the Integrated Project 1 (IP 1), we need additional regularization terms.[1] I.e. given an MPC horizon of T_{MPC} seconds and a time step $t_0 \in [0, T - T_{\text{MPC}}]$ we solve the following optimization problem

$$\begin{aligned}
\min_{\substack{(\mathbf{q}_t)_{t\in[t_0,T_{\text{MPC}}]},\\ (\mathbf{u}_t)_{t\in[t_0,T_{\text{MPC}}]}}} \quad & \mu_{\mathbf{x}} \int_{t_0}^{T_{\text{MPC}}} \left\| \mathbf{x}_t^{\text{tip}} - \mathbf{x}_t^{\mathsf{PP},\text{tip}} \right\| \mathrm{d}t \\
& + \mu_\theta \int_{t_0}^{T_{\text{MPC}}} \mathbb{1}_{|\theta_t| \geq |\theta_t^{\mathsf{PP}}|} \cdot \|\theta_t\| \,\mathrm{d}t \\
& + \mu_{\mathbf{u}} \int_{t_0}^{T_{\text{MPC}}} \|\mathbf{u}_t\| \,\mathrm{d}t \\
& + \mu_{\mathbf{q}} \int_{t_0}^{T_{\text{MPC}}} \|\mathbf{q}_t - \mathbf{q}_{\text{rest}}\| \,\mathrm{d}t \\
& + \mu_{\dot{\mathbf{q}}} \int_{t_0}^{T_{\text{MPC}}} \|\operatorname{diag}(w_{\text{vel}}) \cdot \dot{\mathbf{q}}\| \,\mathrm{d}t \\
\text{s.t.} \quad & \forall t \in [t_0, T_{\text{MPC}}] : \ \mathbf{u}_t \in [\mathbf{u}_{\min}, \mathbf{u}_{\max}], \\
& \mathbf{u}_t = M(\mathbf{q}_t)\ddot{\mathbf{q}}_t + c(\mathbf{q}_t, \dot{\mathbf{q}}_t) + g(\mathbf{q}_t), \\
& \mathbf{x}_t = f_{\text{F}}(\mathbf{q}_t),
\end{aligned} \tag{1}$$

with regularization coefficients $\mu_{\mathbf{x}}, \mu_\theta, \mu_{\mathbf{u}}, \mu_{\mathbf{q}}, \mu_{\dot{\mathbf{q}}} \in \mathbb{R}$ and a custom weighting vector $w_{\text{vel}}^{\mathsf{T}} = (1,1,1,1,0,0)$ for the robot's joint velocities. We use the standard dynamics notation with the mass matrix $M(\mathbf{q})$, the coriolis and centripetal forces $c(\mathbf{q}, \dot{\mathbf{q}})$, the gravity term $g(\mathbf{q})$ and the forward kinematics mapping f_{F}.

C. Experimental Setup

Internally, we use the Pinocchio [12] based optimal control library Crocoddyl[2] [13] to plan trajectories and compute torques. For more information on the proposed Feasibility Driven Differential Dynamic Programming (FDDP) solver, we refer to [14] respectively [4] for a brief summary. To simulate separated solution and execution processes, as on the real system, we use the MuJoCo [15] simulation environment to execute the computed torques. Since our real system is running with 500 Hz, we also use the same control frequency in MuJoCo and thus use an integration time step of $\Delta_t^{\text{MJ}} = 0.002\,\text{s}$.

Based on our results of the IP 1, we use a Barrett Whole Arm Manipulator (WAM)[3] robot with a upwards rotated base plate and a starting configuration vector of $(0, -1.3, 0, 1.3, 0, 0)^{\mathsf{T}}$ corresponding to the standard configuration stated in [4].

In contrast to the pendulum used in the previous experiments, consisting of a 5.5 cm connection pole, two hinge joints modeling the inverted pendulum and a pole of 30 cm, we now have a 3D-printed version (cf. fig. 1) of the two hinge joints and a pendulum pole with a length of 30 cm. We use a replica of the 3D-printed version in our planning and simulation environments.

[1]All cost terms are motivated and explained in detail in Section II.

[2]Code and documentation are available under `https://github.com/loco-3d/crocoddyl`.

[3]For more details see `https://advanced.barrett.com/wam-arm-1`.

II. Simulation Results (Integrated Project 1)

A. Usage of Model Predictive Control

As already discussed in the Robot Learning lecture and in the previous sections, all DDP-based algorithms share the disadvantage of providing valid controls only along the internally computed nominal trajectory. Thus, it is impossible to react to disturbances and compensate for modelling errors between the internal model and the simulation respectively the real system. One solution provided in the lecture to overcome this problem is the usage of a Model Predictive Control (MPC) control law. Since we are working with an unstable system with nonlinear dynamics, the choice of an FDDP based MPC controller seems justified.

B. Evaluation via Setpoint Experiments

In order to evaluate the performance of our controllers, we designed benchmark tests based on *setpoint reaches*. As the name suggests, we want the robot to reach a setpoint relative to its pendulum's tip while balancing the pendulum. Every setpoint reach is defined by a 3-tuple (direction, distance, orientation) and describes a motion in a two-dimensional hyperplane. We use three different directions $\{x, y, z\}$, three different distances $\{0.05\,\mathrm{m}, 0.1\,\mathrm{m}, 0.15\,\mathrm{m}\}$ and two different orientations $\{-1, 1\}$, e.g. the tuple $(x, 0.15, -1)$ corresponds to the task of moving the pendulum's tip to the position $x^{\mathrm{tip}}_{t=0} + (-0.15, 0, 0)^{\mathsf{T}}$. Consequently, enumerating all the different combinations yields 18 different tasks in total.

For each experiment we fix a time-independent target trajectory of 3000 cost nodes, i.e. setpoint reaches with a duration of $3\,\mathrm{s}$ per target. For the evaluation presented in [4], we use the last 300 nodes, i.e. 10% of the trajectory to compute the average positional and rotational error.

C. MPC Horizon and Integration Factor

We obtain an initial benchmark by starting with the naive idea of formulating the setpoint reach as a raw task space optimal control problem. We experiment with MPC horizons of $\{10, 15, 20\}$ cost nodes. When planning with a control frequency of $500\,\mathrm{Hz}$ and a MPC Horizon of 20 cost nodes, the controller's effective planning horizon is limited to $\Delta^{\mathrm{MJ}}_t = 0.002\,\mathrm{s} \cdot 20 = 0.04\,\mathrm{s}$. Since on average, for real-time applications, the computation time of one MPC call can exceed $0.04\,\mathrm{s}$, we introduce an integration factor as a hyper parameter, in order to increase the controller's effective planning horizon.[4]

While increasing the effective planning horizon, a higher integration time decreases the effective control frequency. I.e. by using an integration factor of $\mathsf{f}_{\mathrm{MPC}} = 4$ with a MPC horizon of $\mathsf{h}_{\mathrm{MPC}} = 20$ cost nodes, we obtain an integration time step of $\mathrm{d}t = 4 \cdot \Delta^{\mathrm{MJ}}_t = 0.008\,\mathrm{s}$ with an effective control frequency of $125\,\mathrm{Hz}$ and an effective MPC planning horizon of $0.16\,\mathrm{s}$.

In [4] we show that higher integration factors do not only significantly decrease the computational complexity, but also simultaneously increase the controller's performance. Thus, the tradeoff between a longer planning horizon and a lower real-time control frequency is favorable for the longer planning horizon.

D. Preplanning with the FDDP solver

Although the trajectories we obtain by the raw task space controller are in general smooth and especially the rotational errors are impressively low, the controller's torques are not physically optimal.

The raw task space MPC controller seems to have problems with the tradeoff between reaching the setpoint as fast as possible while keeping the pendulum balanced. Here, the necessary high penalties on the rotation do not enable tipping movements as exemplarily shown in [4, Fig. 2].

When using the FDDP solver for our setpoint problem and visualizing the result (under the assumption of a perfect model), we obtain physically optimal trajectories. Hence, we use Crocoddyl's FDDP solver to preplan an optimal trajectory and follow the resulting trajectory with our MPC controller. Since this computation can be performed offline, we do not limit the real time applicability.

As stated in [4], following the preplanned trajectory with the MPC controller yields three problems that we each solved individually.

First, the trajectories generated by the FDDP solver are extremely aggressive and very similar to bang-bang control, including e.g. bumps in the acceleration. Thus, it is quite difficult to follow the trajectories in simulation and (probably) impossible on the real system. We solved this issue by smoothing the preplanned trajectories via an additional velocity penalty in the joint space to obtain less aggressive tipping movements.[5]

Second, following the trajectory in the state space, i.e. the joint space, is extremely difficult. We either need complex additional penalties on the rotation or we cannot follow the trajectory directly as even small deviations in the joints modeling the pendulum immediately cause a collapse of the system from which we cannot recover. An elegant solution to overcome this problem is to extract the movement of the pendulum's tip in the task space during the offline phase and follow the task space trajectory with the MPC. Additionally, inspired by gain scheduling in control engineering, we use a clipped quadratic barrier penalty function to allow tipping movements similar to the preplanned movement.

Third, the precomputed trajectories consistently exhibit a sinking behavior when the controller is close to the global end of the time horizon. This problem is caused by the controllers myopic behavior to slightly increase the positional penalties in order to fully eliminate the costs on the torques by dropping the end effector. We fix this problem by cutting off the precomputed task space trajectory after 1500 steps, i.e. $1.5\,\mathrm{s}$, and mirror the last point.

[4]A comprehensive list of all hyperparameters and their effects on the system is available in the appendix under table III.

[5]For more complex trajectories we do not expect the necessity of a velocity penalty, as movements inherently encode a velocity and an acceleration.

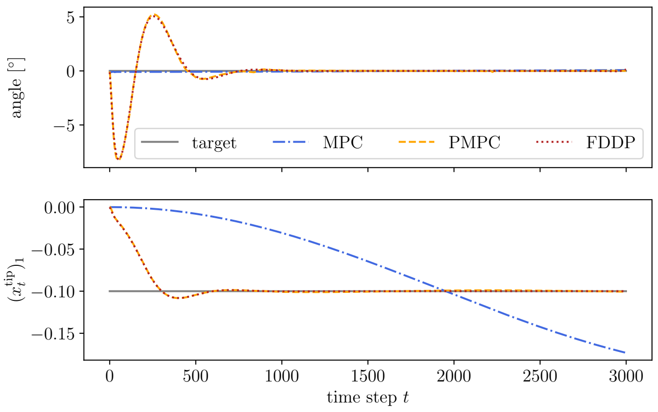


Fig. 2: Comparison of the results for the MPC controller presented in section II-C, the precomputed FDDP trajectory and the resulting trajectory of the preplanned MPC (PMPC) controller, all for the setpoint experiment $(x, 0.1, -1)$. We do not plot the absolute rotational error, but instead show the angle between the orientation vector projected onto the xz-plane with the (projected) z-axis. Plots for the y- and z-axis are not shown since the errors are on average below 10^{-3}. A comprehensive comparison of the results is shown in table I in the appendix.

E. *Evaluation of Setpoint Experiments*

All results are available in full detail under [here]. Furthermore, we provide videos comparing the raw task space MPC controller and the preplanned MPC (PMPC) controller under [here]. Overall, the PMPC controller clearly outperforms the raw task space MPC controller and nearly perfectly matches the preplanned trajectory computed by Crocoddyl's FDDP solver. As expected, the MPC controller with the biggest planning horizon of 20 time steps achieves the best results. The positional errors being in the order of magnitude 10^{-4} mean that we are in the millimeter range. By comparing the positional error of the MPC with the positional error of the FDDP solution, we observe that our controller is impressively close to the precomputed ground truth. In general, the rotational errors are close to zero degrees, so we can see that the controller is able to perfectly balance the pendulum after the initial desired tipping movement.

F. *Trajectory Following*

In our final experiment, we generalize our results to time-varying trajectories: a circular movement, being embedded in the two-dimensional xz-hyperplane, and a spiral movement, representing a full three-dimensional movement. As shown in [4], we keep the parameters fixed, except for some small penalty parameters adjustments on the joint velocities and removing the velocity penalty in the preplanning. As before, videos of the controller following the two trajectories are available under [here].

In conclusion, the results in simulation are really astonishing as the proposed method is able to follow quite complex trajectories at a high frequency with very little error.

III. Simulation Results (Integrated Project 2)

In order to transfer our results to the real system, we need to replicate the real system's processes as realistically as possible to minimize the Sim2Real gap. We identified four major differences between our initial implementation and the implementation on the real system:

1) The existence of time delays due to parallel execution of solver computations and the robot's execution.
2) The usage of *OptiTrack*[6] instead of absolute joint encoders to estimate the pendulum's joint positions.
3) Friction, damping and armature effects, which we neglected in the first part of the project, but now have to be taken into account.
4) Imperfect observations caused by noisy joint position measurements and uncertain pendulum positions as well as lower frequency updates of the pendulum compared to the robot.

In the following paragraphs we will present, how we approach each of the aforementioned problems and what type of solution proved to be suitable. To simplify our task, we started to work on a stabilization task, i.e. a setpoint goal of $(x, 0.0, 0)$.

A. *Simulation of Time Delays*

Caused by parallel execution of the solver's torque computation and the robot's torque execution, three major problem arise. First, as shown in fig. 3a, given a solver computation time of n steps (in $500\,\text{Hz}$), the first n torques of the solution trajectory are omitted since every torque is internally mapped to a corresponding time stamp. In our experiments, we observe that especially the first five to seven torques of each MPC-solution are essential in stabilizing the pendulum (since these are used aggressively to achieve the desired movement). Thus, skipping these torques immediately causes system crashes.

Second, due to differences between the solver-internal model and the real system, the states $\bar{s}_n$ and s_n differ significantly and also lead to unstable execution of the torques $\mathbf{u}_n, \mathbf{u}_{n+1}, \ldots$, since the real state of the system and the corresponding internally planned state do not match anymore.

Third, we could not use the previous solver output as a warmstart anymore, as the solution trajectory is significantly different from the previous solution. This issue was easily solvable by using the Crocoddyl internal *quasi-static* warmstart.[7]

We solved problem one and two by using a state observer with a Pinocchio-based forward integration to estimate the system's state *after* the MPC computation and use this state $\tilde{s}_n$ as the solver input (cf. fig. 3b). By tuning the simulation and the solver-internal model, we are able to reduce the

[6]See `https://optitrack.com/`.

[7]Crocoddyl offers the option of a *quasi-static* warmstart, which tries to stabilize a given reference posture by setting the acceleration equation to zero and solving the resulting nonlinear equation system using Newton's method.

problem of non-matching state estimates to a negligible difference.[8]

B. Replicate OptiTrack Observations

In contrast to the controller scheme presented in [10], we cannot rely on absolute joint encoders to compute the pendulum's joint positions. Instead, we are using the motion capture system OptiTrack to estimate the pendulum's position and orientation in space. On the real system, we have four markers placed on the pendulum pole. OptiTrack will return a three-dimensional vector in the world-frame for each marker, replicated in our simulation by four MuJoCo-sites. In order to extract the pendulum's orientation and recover the configuration of the two underlying joints, we

1) Perform a linear regression with the yz-coordinates as inputs and the x-coordinates of the observations as targets, to filter out noise and measurement uncertainties, yielding a unit vector with the direction of the pendulum in the world frame,[9]
2) Transform the pendulum's direction vector back into the local end-effector frame, to ensure invariance under rotations along the z-axis, and
3) Use trigonometric identities, to recover the joint positions modeling the pendulum.

In order to compute the pendulum's joint velocities, we started with a forward difference computation, with an additional low-pass filter (see section III-D for more details). Since we have with $120\,\mathrm{Hz}$ a lower update frequency compared to the robot, we use a higher time difference when computing the finite difference derivatives.

C. Friction, Armature and Damping: Custom Crocoddyl-Model

As mentioned before, in the first part of the Integrated Project, we neglected all friction, damping and armature effects for simplicity. Since the real system of course exhibits all these effects, we need to adjust our model and simulation accordingly.

Thanks to Pascal, we provide a custom Crocoddyl friction model with optional gravity compensation in Python and C++ and also well-tuned friction, damping and armature parameters, obtained by running a system identification on the Barrett WAM in the IAS laboratory.

For damping and armature, we use the standard computation which is also implemented in MuJoCo. In contrast to MuJoCo, Crocoddyl is not able to perform implicit computations to (correctly) represent coulomb friction.[10] Thus, we are using a $\tanh$-Friction Model, as stated in [16], with a slope of $\gamma = 100$.

When working with the custom model, we observe that the dynamic model implemented in Python decreases Crocoddyl's computation enormously compared to the C++ version. While the preplanning without our custom model took on average $1.8\,\mathrm{s}$, the Python implementation slowed it down to an average of $28\,\mathrm{s}$, which is an increase of factor 15. The same holds true for the MPC computation, although the time increase is smaller due to less cost nodes.

More importantly, Crocoddyl has significant troubles when working with friction. Not only are the solver iterations slower, but more essentially the number of iterations to solve the optimal control problem increases from less than 100 iterations to more than 3000 iterations for the preplanning of the stabilization task. This behavior can be explained by the relatively unstable friction modeling. As the solver has a big integration time step compared to the $\tanh$-slope, the solver perceives the friction as a step-function, causing long computational times to stabilize this behavior.

Our solution to this problem is to exclude coulomb friction in the planning and only model damping and armature in the FDDP solver. Naturally, this causes the MPC computation to be less accurate, but our experiments show that the tradeoff between slight inaccuracies and a significant decrease in computational complexity, is favorable for not modeling friction effects in the FDDP solver.[11] Since friction does not have a negative impact on the computation time of the forward integration, we use the complete friction model in the state observer to obtain better state estimates. Additionally, we experiment with compensating for friction losses by adding the negative output of the friction model to the solver's torque outputs. This causes significant instabilities and a zig-zag behavior of the robot, which is why we ultimately do not compensate for friction losses at all.

D. Noise on Joint Positions and OptiTrack Observations

As stated in the beginning of this chapter, one of the biggest challenges in the transition from the simulation to the real system are imperfect measurements and noisy joint observations. To replicate the real system, we attach our simulation observations with uniform noise to test our filtering methods and examine how robust the FDDP solver is with respect to noise.

Since we cannot rely on absolute joint encoders, reliably computing the joint velocities for the pendulum's joints is extremely challenging due to noise amplifications when working with derivatives of noisy measurements.

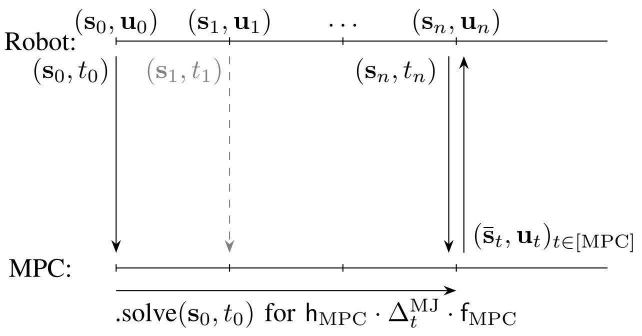
1) Low-pass Filter for FD-Velocities: In a first step, we adjust our state observer to not observe the joint velocities anymore, but instead only observe the MuJoCo joint positions with additional uniform noise. Like on the real system, we reimplemented the same low-pass filter that is running on the Barrett WAM, to remove high-frequency noise in

[8]Without noise and friction, we achieved average ℓ^2-differences of $1 \cdot 10^{-4}\,\mathrm{m}$ and lower between $\mathbf{s}_n$ and $\tilde{\mathbf{s}}_n$.

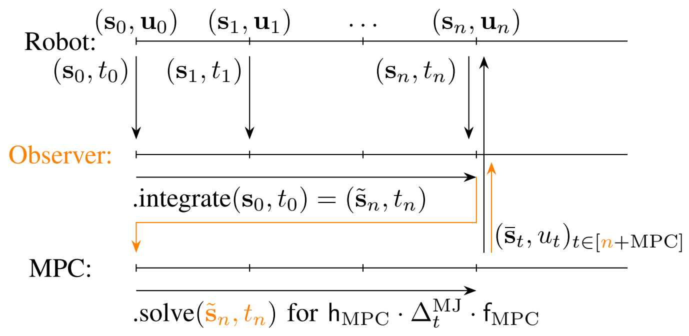
[9]This methodology fails, if the x-coordinates of all four observation vectors are numerically equal, which corresponds to a joint angle of 90° and thus a fully tipped pendulum which usually corresponds to a crash of the system.

[10]We see this deviation in friction modeling as an opportunity to test the robustness of our algorithms, since MuJoCo is also not fully modeling the friction effects of the real system.

[11]Since Crocoddyl internally mostly relies on computing the derivatives of the dynamics model to provide a backward pass in the solver, neglecting coulomb friction can be justified by the fact that the derivatives of the coulomb-friction are either zero (for any point $x \neq 0$) or undefined (for the discontinuity at $x = 0$).


Robot:
$(\mathbf{s}_0, \mathbf{u}_0)$
$(\mathbf{s}_1, \mathbf{u}_1)$
$\dots$
$(\mathbf{s}_n, \mathbf{u}_n)$
$(\mathbf{s}_0, t_0)$
$(\mathbf{s}_1, t_1)$
$(\mathbf{s}_n, t_n)$
$(\bar{\mathbf{s}}_t, \mathbf{u}_t)_{t\in[\mathrm{MPC}]}$
MPC:
$.\mathrm{solve}(\mathbf{s}_0, t_0)$ for $\mathsf{h}_{\mathrm{MPC}} \cdot \Delta_t^{\mathrm{MJ}} \cdot \mathsf{f}_{\mathrm{MPC}}$

(a) Simulation of time delays without a forward integrator. The MPC controller receives a state observation at time step t_0 and we simulate a computation time of n steps in 500 Hz. The resulting solution trajectory consists of states $(\bar{\mathbf{s}}_t)_{t\in[\mathrm{MPC}]} := \{\mathbf{s}_0, \bar{\mathbf{s}}_1, \dots, \bar{\mathbf{s}}_{\mathsf{h}_{\mathrm{MPC}}\cdot\mathsf{f}_{\mathrm{MPC}}}\}$ and corresponding torques. Thus, since the real system proceeds to time step t_n during the solver's computation, the first n torques of the preplanned trajectory are **omitted**. Additionally, all intermediate updates published by the real system during the solution process are lost and cannot be used to improve the state estimate.


Robot:
$(\mathbf{s}_0, \mathbf{u}_0)$
$(\mathbf{s}_1, \mathbf{u}_1)$
$\dots$
$(\mathbf{s}_n, \mathbf{u}_n)$
$(\mathbf{s}_0, t_0)$
$(\mathbf{s}_1, t_1)$
$(\mathbf{s}_n, t_n)$
Observer:
$.\mathrm{integrate}(\mathbf{s}_0, t_0) = (\tilde{\mathbf{s}}_n, t_n)$
$(\bar{\mathbf{s}}_t, u_t)_{t\in[n+\mathrm{MPC}]}$
MPC:
$.\mathrm{solve}(\tilde{\mathbf{s}}_n, t_n)$ for $\mathsf{h}_{\mathrm{MPC}} \cdot \Delta_t^{\mathrm{MJ}} \cdot \mathsf{f}_{\mathrm{MPC}}$

(b) Simulation of time delays with a Pinocchio-based forward integrator. The state observer performs a forward integration for a total time horizon of n steps in 500 Hz (estimated solver time), yielding an estimate $\tilde{\mathbf{s}}_n$ of the system's state at time step t_n given the current state and the list of torques. Consequently, the solver's output consists of states $(\bar{\mathbf{s}}_t)_{t\in[n+\mathrm{MPC}]} := \{\mathbf{s}_n, \bar{\mathbf{s}}_{n+1}, \dots, \bar{\mathbf{s}}_{n+\mathsf{h}_{\mathrm{MPC}}\cdot\mathsf{f}_{\mathrm{MPC}}}\}$ and corresponding torques. Thus, the real system will **not omit** the very first and most important torques which are required to stabilize the pendulum. As a side effect, the state observer also allows to cache all intermediate state updates to improve the system's state estimate also during a solver call.

Fig. 3: Schemes to visualize parallel solver execution and state updates of the real system with and without a forward integrator to compensate for time delays caused by solver computation. State indices are counted in 500 Hz steps, i.e. the difference between two tick marks is $\Delta_t^{\mathrm{MJ}} = 0.002\,\mathrm{s}$, as on the real system. For simplicity, updates by OptiTrack are not visualized, but could be easily added to the state observer with an update rate of 120 Hz, i.e. nearly every forth tick-mark (cf. fig. 6).

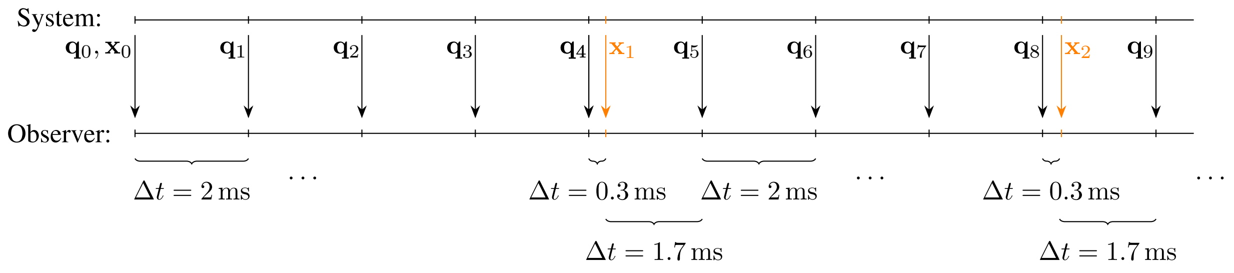

System:
$\mathbf{q}_0, \mathbf{x}_0$
$\mathbf{q}_1$
$\mathbf{q}_2$
$\mathbf{q}_3$
$\mathbf{q}_4$
$\mathbf{x}_1$
$\mathbf{q}_5$
$\mathbf{q}_6$
$\mathbf{q}_7$
$\mathbf{q}_8$
$\mathbf{x}_2$
$\mathbf{q}_9$
Observer:
$\Delta t = 2\,\mathrm{ms}$
$\Delta t = 0.3\,\mathrm{ms}$
$\Delta t = 2\,\mathrm{ms}$
$\Delta t = 0.3\,\mathrm{ms}$
$\Delta t = 1.7\,\mathrm{ms}$
$\Delta t = 1.7\,\mathrm{ms}$

Fig. 4: Internal integration scheme for the Luenberg observer. We obtain joint position updates of the robot at a frequency of 500 Hz, indicated by the black tick-marks on the time line. Each joint position update $\mathbf{q}_t \in \mathbb{R}^4$ only contains information about the four DoFs of the Barrett WAM. Since OptiTrack only provides updates at 120 Hz (orange tick marks), the pendulum position updates $\mathbf{x}_t := (\mathbf{x}_t^1, \dots, \mathbf{x}_t^4) \in \mathbb{R}^{3\times3\times3\times3}$, which are converted to pendulum joint positions (cf. section III-B), cause uneven integration time steps. Each integration time step is computed as the difference between the time step of the last observation and the time step between the current observation. If at an integration step no observation of the robot or the pendulum is given, we use zero gains for the robot respectively the pendulum in the nonlinear observer. Due to network and communication processes, each of the above stated updates can have a small additional delay.

the finite difference derivative computation, caused by noisy measurements.

In a second step, we also perturb each OptiTrack observation vector by adding a random unit vector scaled with a uniform noise factor, since we expect to have measurement errors up to 1 cm on the real system. Although we use a linear regression on the four measurements, the reconstructed joint positions are still noisy enough to let the finite difference computation of the joint velocities crash the controller.

Initially, we tried to use the same low-pass filter as on the Barrett WAM with a higher cut-off frequency, but we were not able to stabilize the pendulum using this filter. This can most probably be explained by the high eigenfrequency of the system, which requires either a lower cut-off frequency, causing a significant phase-shifts between the filtered and the correct derivatives, or too noisy derivatives when using a higher cut-off frequency.

2) Nonlinear State Observer: Additionally, the pendulum dynamics are highly dependent on the robot's state and applied torques, which are not incorporated in a naive low-pass filter. To inherently include the system dynamics, we implement a Luenberg state observer [17, Sec. 3.3.2] for the

WAM and the pendulum to provide a state estimate based on the system dynamics model (via Pinocchio) and additional feedback gains yielding a weighted average of the model-based state estimate and the actual state observation.

As for the naive low-pass filter, the standard Luenberg observer has the disadvantage of requiring joint velocity estimations, which in our case are highly noisy and instable, caused by unsteady finite difference computations. Consequently, the standard Luenberg observer also leads to unstable and hard shaking behavior of the controls and in many cases also to system crashes.

Apart from demonstrating the well-known difficulties of estimating derivatives of noisy measurements, our experiments with different observers also point out, how sensitive the FDDP solver is with respect to the (estimated) input state. As previously demonstrated in section III-A, when simulating time delays, already slight offsets or inaccuracies in the state estimate will result in zig-zag torques, jumps in the accelerations or even system crashes, as the solver is really aggressive and thus not sufficiently robust.

3) Extended Nonlinear State observer: For the purpose of not relying on unsteady derivatives, we switch to an extended Luenberg observer, which only requires the joint position estimations and not the joint velocities anymore. Compared the the standard Luenberg observer, the feedback gains (for position, velocity and acceleration disturbance) are now entirely based on the difference between the internally simulated joint positions and the observed joint positions. Again, internally the observer relies on the full friction model including coulomb friction.

In order to obtain optimal feedback gains, we implement a hyper parameter search by tracking a perfect trajectory with additional noise in simulation, allowing us to perform the stabilization task.

4) FDDP Solver Configuration: Besides using more complex filtering operations and nonlinear state observers, we also experiment with different FDDP solver configurations. As already mentioned in section III-A, we switch back to quasi static warmstarts. More importantly, when working with noise amplitudes of 10^{-3} and higher, we observe that the FDDP solver provides in general really aggressive solution trajectories.[12] While aggressively optimal trajectories are justified in the preplanning, we need to bound the maximum number of iterations to 20 in order to ensure robustness of our controller. This upper bound was determined experimentally, since higher bounds provided either unstable or crashing solutions in various experiments and lower bounds did not allow the solver to converge properly. Nevertheless, as also mentioned in section III-D.2, we still have problems with overly aggressive MPC torques causing unstable behavior.

E. Stabilization Analysis, Trajectory Following and Conclusion

Under [here] we provide a video of the experiment shown in fig. 5 as well as the corresponding experiment in the Integrated Project 1 setting. By analyzing fig. 5, we can clearly see that even little error arising from noise and friction effects propagate to significant errors in the state estimate, ultimately causing an unstable controlling behavior and a tracking error increase by factor 106 compared to the last semester's experiment. As previously mentioned, this is most likely caused by very aggressive torques obtained by the MPC paired with the fact that the controller is unable to react to disturbances during the computation time of the MPC.

[12]This matches our findings of section II-D when working with physically optimal trajectories.

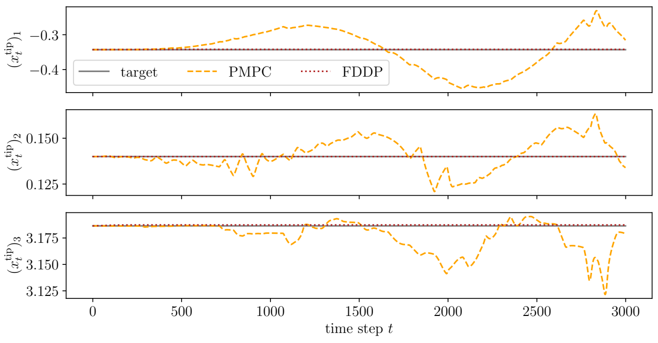


Fig. 5: Task space results for the stabilization task with the final MPC controller scheme presented in section III with 10^{-3} noise on the pendulum observations $x_1, \ldots, x_4$ and $5 \cdot 10^{-4}$ noise on the joint observations q_{wam}. We limit the maximum number of iterations of the solver to 20 and use a forward prediction horizon of 10 steps, i.e. a simulated MPC computation of $0.02\,\text{s}$. The average ℓ^2-difference between the observed state $\tilde{\mathbf{s}}_n$ and the real state $\mathbf{s}_n$ (without noise) amounts $0.044\,\text{m}$ and the average tracking error between $\mathbf{x}^{\text{tip}}$ and $\mathbf{p}^{\text{tip}}$ is $4.933 \cdot 10^{-2}\,\text{m}$. For more a more detailed analysis see section VI-C.

Ensuing from this unstable behavior, we were unfortunately not able to implement a trajectory following controller scheme which is able to compensate for noise, friction and delays.

IV. Hardware Experiments

A. Hardware Setup

To ensure maximum performance and to easily interact with the local hardware setup in the IAS laboratory, reimplementing our code in C++ was required. As shown in fig. 6, we provide C++ implementations for a general control law interface, the Luenberg Observer and helper functions, to e.g. reconstruct the pendulum's joint angles and perform a forward integration.[13]

B. Linear Feedforward Controller with PD-Feedback (LQR)

Since our two main problems are too aggressive torques provided by the MPC controller and the instability to react to disturbances during the MPC computation, we decided to run a experiments with a simplified setting, by generating a *Linear Feedforward Controller* with PD-feedback based

[13]The entire code is available under [here].

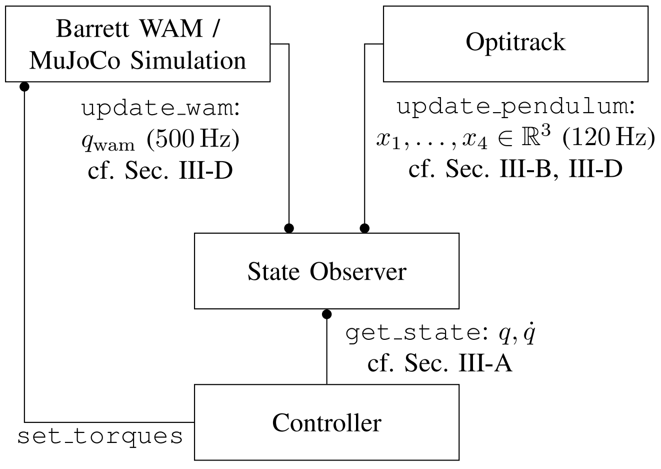

Barrett WAM / MuJoCo Simulation
Optitrack
update_wam: q_{wam} (500 Hz) cf. Sec. III-D
update_pendulum: $x_1, \ldots, x_4 \in \mathbb{R}^3$ (120 Hz) cf. Sec. III-B, III-D
State Observer
get_state: $q, \dot{q}$ cf. Sec. III-A
Controller
set_torques

Fig. 6: Generic controller schema of the real system including joint positional updates of the Barret WAM at 500 Hz and OptiTrack positional updates at 120 Hz.

(LQR controller) on Crocoddyl's internal intermediate results when linearizing the dynamics.

The full derivation of the controller scheme is presented in section VI-D respectively in [4, Appendix, Section B]. In contrast to the MPC control law, the feedforward controller does not require a friction model and no online computation time, i.e. the control matrices and linear offsets can be computed during the offline phase and afterwards each computation just consists of one matrix multiplication, and simultaneously allows to react to disturbances via the PD control law for feedback. Thus, we also do not require a feedforward integrator to compensate for computational time delays. Additionally, as shown in table V the cost models and thus also the tuning of penalty simplifies. As already mentioned in section II-A, this type of controller has the downside that the feedforward torques are only valid along the nominal trajectory.

1) Results in Simulation: A full analysis of the results in simulation is available in section VI-D.2. In fig. 7 and fig. 8 we provide a detailed visualization of the stabilization task respectively the trajectory following task.

In conclusion, the linear feedforward controller with PD-feedback, provides surprisingly convincing results and allows to react to high perturbations while not crashing the system. Futhermore, the computational time is completely offline which is a major advantage for a real-time application. Based on the results in simulation, we decide to bring the LQR controller to the real system.

C. Hardware Experiments

1) Tuning Observer Gains and Control Penalties: When performing the trajectory tracking task, we observed that the K_k matrices derived in eq. (2) have a high order of magnitude and thus cause system crashes when dealing with noise and delays. As shown in fig. 9 we obtain strong torque oscillations caused by high feedback gains. Thus, we had to increase the control penalties to decrease the magnitudes in the feedback matrices and simultaneously increase the penalty on the resting position to ensure a stable trajectory tracking. One can clearly see that the torques applied to the real system are still noisy and have a higher magnitude than the torques computed in simulation.

Furthermore, we tuned the Luenberg observer gains to correctly filter out noise and ensure a stable state estimate, since we only run our controller at 120 Hz. On the real system, we use parameters of $L_{\text{P}} = 0.25$, $L_{\text{D}} = 25$ and $L_{\text{I}} = 0.0$. Internally, the observer uses the full friction model.

2) Evaluation: The full results are shown in fig. 10 respectively fig. 11 and we provide videos under [here].

Overall, we achieve convincing results and we are surprised by how well a simple linear feedforward controller with PD-feedback performs. For the stabilization task we achieve an average error of $4.9225 \cdot 10^{-4}$ m over a duration of 40 s and for the trajectory tracking the LQR controller achieved an average error of $3.2270 \cdot 10^{-2}$ m over a time period of 45 s.

V. Conclusion and Current Work

In conclusion, our analysis and experimentation highlight that the simpler Linear Quadratic Regulator (LQR) controller emerges as a preferred choice over the Model Predictive Control (MPC) approach for several reasons. The LQR controller offers streamlined gain tuning, reducing the complexity associated with controller design. Additionally, its lack of time delays eliminates the need for complex state estimation methods to compensate for potential time losses resulting from parallel execution. The LQR's ability to promptly respond to state disturbances is advantageous for scenarios requiring quick adjustments. Furthermore, the LQR controller can operate at a control frequency of 125Hz, contributing to real-time performance.

Nevertheless, our experiments emphasize the significance of careful considerations when opting for the LQR approach. Achieving optimal controller performance demands meticulous tuning of feedback gains, underscoring the need for a thorough design process. Similarly, selecting appropriate observer gains is essential to ensure accurate and credible state estimates.

In essence, while the LQR controller offers distinct advantages in terms of simplicity and responsiveness, its successful implementation necessitates meticulous gain tuning and observer selection. These findings underscore the importance of a comprehensive evaluation of trade-offs when making controller decisions for practical applications.

Acknowledgment

Ultimately, I would like to thank my two outstanding supervisors, not only for this amazing project they came up with, but especially for all of our fruitful discussions, your creative ideas and endless motivation to experiment with new solution ideas. Thank you very very much!

VI. APPENDIX

A. Full Results of the Shown Experiments

The comprehensive results of the examples shown in fig. 2 and section II-F are presented in table I respectively table II. All results of the Integrated Project 1 are available under [here].

B. Hyperparameters used in the MPC solver

A comprehensive list of all hyperparameters used in simulation and for the FDDP solver is shown in table III.

C. Stabilization Experiment: Noise vs. without Noise

The solver configurations for the experiments shown in fig. 5 are listed in table IV. In both experiments we used a MPC horizon of 20 steps and an integration factor of 8.

The graph in fig. 5 shows an average ℓ^2-difference between the observed state $\tilde{\mathbf{s}}_n$ and the real state $\mathbf{s}_n$ (without noise), of $0.044\,\mathrm{m}$ where the average differences between the joint positions amounts $0.003\,\mathrm{m}$ and for the joint velocities $0.083\,\mathrm{m}$.

The average tracking error between $\mathbf{x}^{\mathrm{tip}}$ and $\mathbf{p}^{\mathrm{tip}}$ is $4.933 \cdot 10^{-2}\,\mathrm{m}$ and thus increased by factor of 106 compared to an average tracking error of in the IP 1 setting.

D. Linear Feedforward Controller with PD-Feedback (LQR controller)

1) Theoretical Derivation: Internally, Crocoddyl relies on optimal linear feedback laws, for which we can extract the parameters. Given a successful backward pass, as stated in [4, Appendix, Section B], the forward pass is of the form

$$\mathbf{u}_k^{\mathrm{new}} = \mathbf{u}_k + \delta_u = \mathbf{u}_k + K_k(\mathbf{x}_k^{\mathrm{new}} - \mathbf{x}_k) + k_k, \quad (2)$$

$$\mathbf{x}_{k+1}^{\mathrm{new}} = f(\mathbf{x}_k^{\mathrm{new}}, \mathbf{u}_k^{\mathrm{new}}), \quad (3)$$

with matrices $K_k = -Q_{uu,k}^{-1} Q_{ux,k}$ and vectors $k_k = -Q_{uu,k}^{-1} Q_{u,k}$ for the optimal linear feedback controller at each discrete time step k.

Thus, given a successful solver run, we can extract the derivative matrices and vectors $(Q_{uu,k}^{-1}, Q_{ux,k}, Q_{u,k})_k$ as well as the optimal torques $(\mathbf{u}_t)_t$ to construct a linear feedforward controller.[14] First, we compute the control matrices K_k and the linear offset k_k for each time step k. Next, we directly obtain the linear feedforward (FF) controller via

$$\mathbf{u}_t^{\mathrm{FF}} = \mathbf{u}_t + K_k \cdot \left(\mathbf{s}_k^{\mathrm{d}} - \mathbf{s}_k\right) + k_k \quad (4)$$

given desired states $(\mathbf{s}_k^{\mathrm{d}})_k = \left((\mathbf{q}_k^{\mathrm{d}}, \dot{\mathbf{q}}_k^{\mathrm{d}})\right)_k$ and a time step k.

2) Results: The results for the LQR controller are shown in fig. 7 and fig. 8. In both cases the controller operates at $125\,\mathrm{Hz}$ and the positions as well as the velocities were pertubated with uniform noise to simulate measurement errors.

Interestingly, in the trajectory following example, one can clearly see a slight phase shift between the desired and achieved trajectory in the time interval of $1.5\,\mathrm{s}$ - $2\,\mathrm{s}$. This is also represented in the torques' visualization in fig. 9, predominately for the joints one and three.

[14]Fortunately, Crocoddyl directly provides the inverse matrices to ensure numerical stability and optimal computational performance.

REFERENCES

[1] Bernard Widrow. "Pattern-recognizing control systems". In: *Compurter and Information Sciences* (1964).

[2] Atilla Bayram, Fırat Kara, and M. Almali. "Design of Spatial Inverted Pendulum System". In: vol. 291. Aug. 2019, p. 02004. DOI: 10.1051/matecconf/201929102004.

[3] Ishan Chawla, Vikram Chopra, and Ashish Singla. In: *International Journal of Nonlinear Sciences and Numerical Simulation* 22.2 (2021), pp. 183–195. DOI: doi:10.1515/ijnsns-2018-0029. URL: https://doi.org/10.1515/ijnsns-2018-0029.

[4] Florian Wolf, Pascal Klink, and Kai Ploeger. *Pendulum Acrobatics, Integrated Project 1.* 2023. URL: https://sites.google.com/view/pendulumacrobatics/ip1.

[5] Chi Youn Chung et al. "Balancing of an inverted pendulum with a kinematically redundant robot". In: *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289).* Vol. 1. 1999, 191–196 vol.1. DOI: 10.1109/IROS.1999.813003.

[6] X. Albouy and L. Praly. "On the use of dynamic invariants and forwarding for swinging up a spherical inverted pendulum". In: *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187).* Vol. 2. 2000, 1667–1672 vol.2. DOI: 10.1109/CDC.2000.912101.

[7] G. Schreiber, C. Ott, and G. Hirzinger. "Interactive redundant robotics: control of the inverted pendulum with nullspace motion". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180).* Vol. 1. 2001, 158–164 vol.1. DOI: 10.1109/IROS.2001.973352.

[8] Alonso Marco et al. "Automatic LQR tuning based on Gaussian process global optimization". In: *2016 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, May 2016. DOI: 10.1109/icra.2016.7487144. URL: https://doi.org/10.1109%2Ficra.2016.7487144.

[9] Andreas Doerr et al. "Model-Based Policy Search for Automatic Tuning of Multivariate PID Controllers". In: *CoRR* abs/1703.02899 (2017). arXiv: 1703.02899. URL: http://arxiv.org/abs/1703.02899.

[10] Minh Nhat Vu, Christian Hartl-Nesic, and Andreas Kugi. "Fast Swing-Up Trajectory Optimization for a Spherical Pendulum on a 7-DoF Collaborative Robot". In: *2021 IEEE International Conference on Robotics*

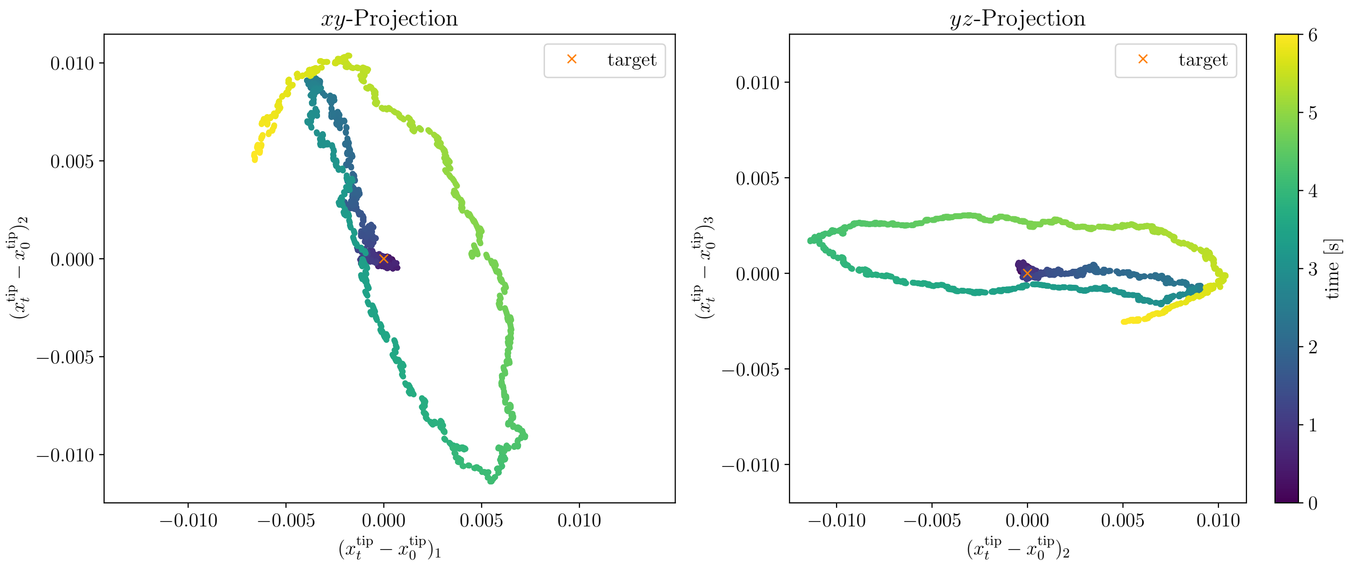


Fig. 7: Projected trajectories of the pendulum's tip position for the stabilization task in simulation, solved with the LQR controller (differences on the axis are to be interpreted setwise). The controller operates at $125\,\text{Hz}$ and the positions and velocities were pertubated by uniform noise with a order of magnitude of $1 \cdot 10^{-3}$. Overall the controller achieved an average positional error for the pendulum's tip of $5.804 \cdot 10^{-4}\,\text{m}$, i.e. less than one millimeter. The corresponding videos are available under [here]. The trace color in the videos is based on the current velocity of the pendulum's tip.

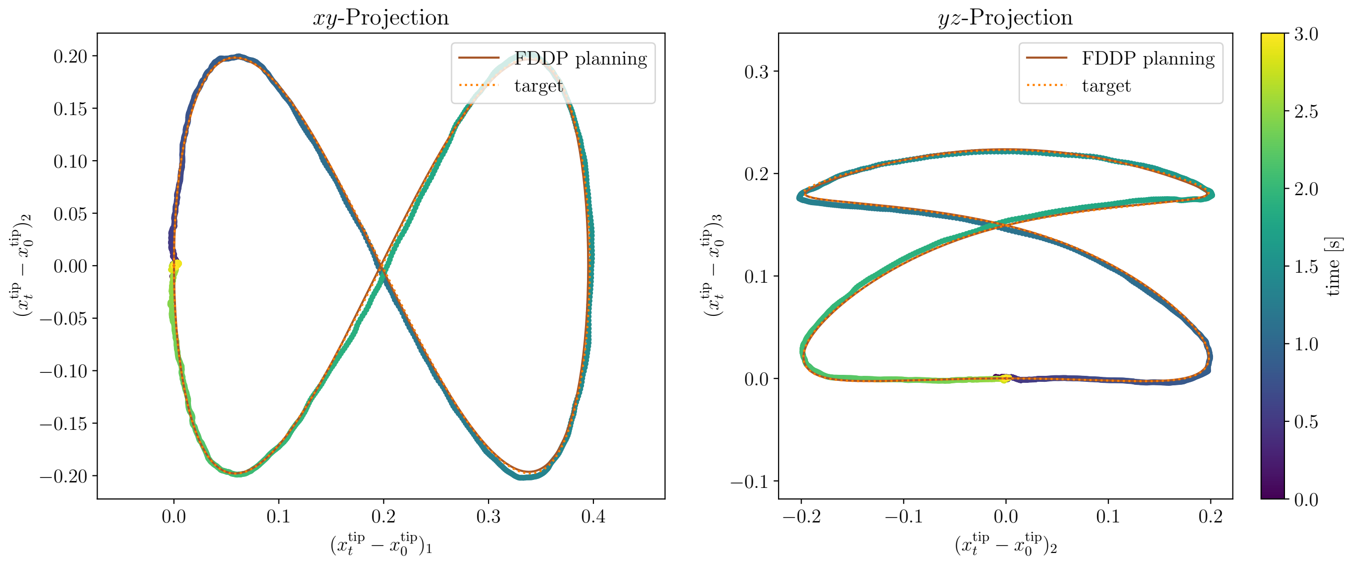


Fig. 8: Projected trajectories of the pendulum's tip position for the trajectory following task in simulation, solved with the LQR controller. The controller operates at $125\,\text{Hz}$ and the positions and velocities were pertubated by uniform noise with a order of magnitude of $1 \cdot 10^{-3}$. Overall the controller achieved an average positional error for the pendulum's tip position of $3.633 \cdot 10^{-3}\,\text{m}$, i.e. also less than a centimeter, while the preplanned trajectory of the FDDP solver achieves an average positional error of $5.483 \cdot 10^{-4}\,\text{m}$, i.e. less than a millimeter. The corresponding videos are available under [here]. The trace color in the videos is based on the current velocity of the pendulum's tip.

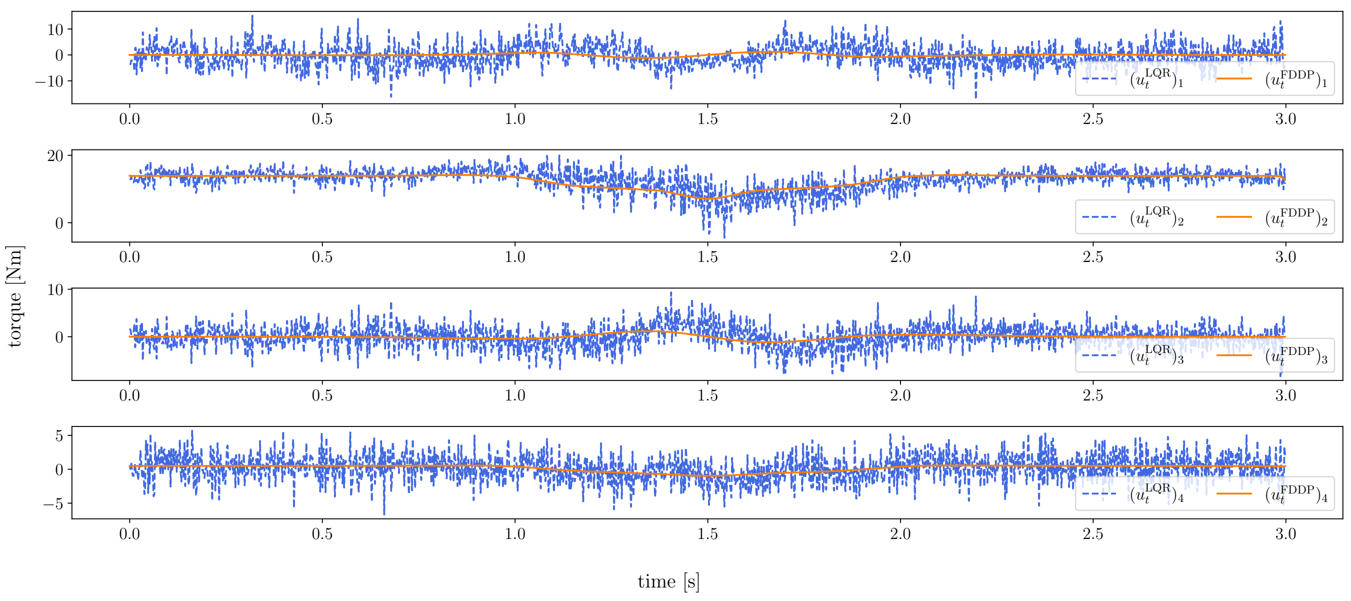


Fig. 9: Torques of the linear feedforward controller with PD-feedback (LQR in the plot) and the internally preplanned FDDP torques for the tractory tracking task (in simulation). Differences on the axis are to be interpreted setwise. The controller operates at $125\,\mathrm{Hz}$ and the positions and velocities were pertubated by uniform noise with a order of magnitude of $1 \cdot 10^{-3}$. Overall the controller achieved an average positional error for the pendulum's tip position of $3.633 \cdot 10^{-3}\,\mathrm{m}$, i.e. also less than a centimeter, while the preplanned trajectory of the FDDP solver achieves an average positional error of $5.483 \cdot 10^{-4}\,\mathrm{m}$, i.e. less than a millimeter. The corresponding videos are available under [here]. The trace color in the videos is based on the current velocity of the pendulum's tip.

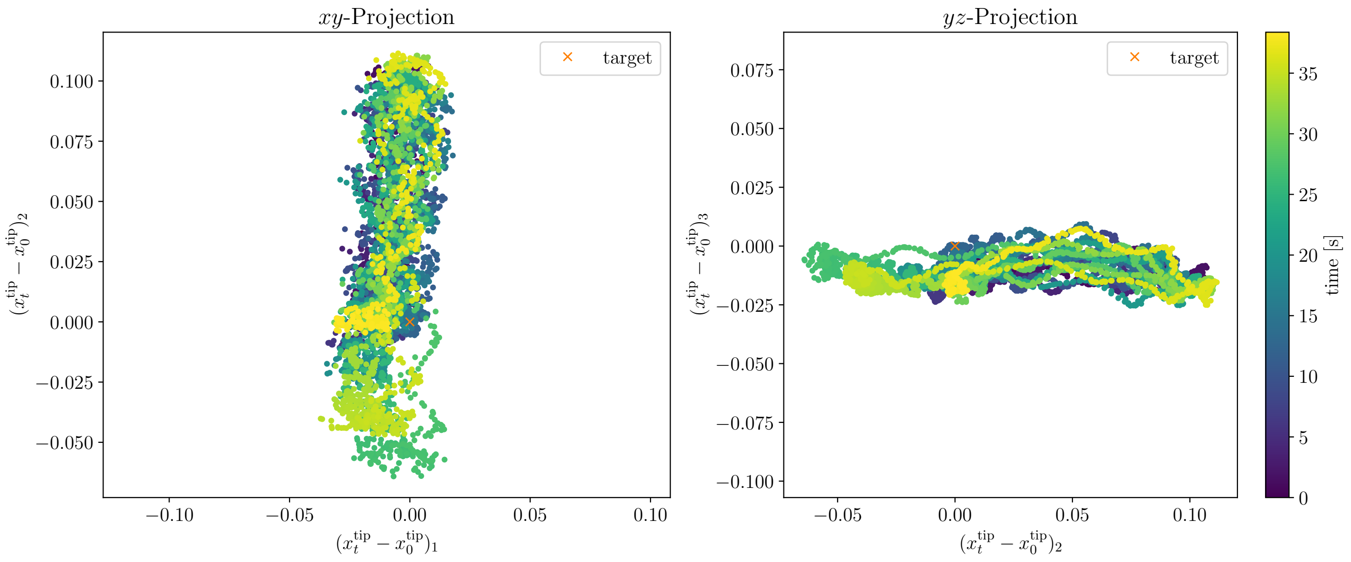


Fig. 10: Projected trajectories of the pendulum's tip position for the stabilization task on the real system, solved with the LQR controller. The controller operates at $120\,\mathrm{Hz}$ and obtains an average tracking error of $4.9225 \cdot 10^{-4}\,\mathrm{m}$. The corresponding videos are available under [here].

	MPC	PMPC	(FDDP Preplanning)
Positional Error [m]	$9.237 \cdot 10^{-02}$	$\mathbf{7.065 \cdot 10^{-03}}$	$6.534 \cdot 10^{-04}$
Rotational Error [deg]	$2.284 \cdot 10^{-01}$	$\mathbf{4.298 \cdot 10^{-02}}$	$7.491 \cdot 10^{-02}$
Computational Time [s]	$7.924 \cdot 10^{+00}$	$\mathbf{6.011 \cdot 10^{+00}}$	$9.704 \cdot 10^{-01}$

TABLE I: Results for the setpoint experiment $(x, 0.1, -1)$ shown in fig. 2.

	Circle	Spiral
Positional Error [m]	$6.32 \cdot 10^{-03}$	$6.77 \cdot 10^{-03}$
Computational Time [s]	$6.89 \cdot 10^{+00}$	$6.87 \cdot 10^{+00}$

TABLE II: Results for the time-varying trajectory experiment shown in section II-F.

Hyperparameter in Simulation	Effect on the Solver and the Solution
MPC Horizon	Number of cost nodes the controller can look ahead.
Integration Factor	Increase the effective MPC horizon by increasing the duration of an action and decrease the effective control frequency.
Cutoff time for preplanned trajectory	Avoid to reproduce the sinking behavior in the preplanned trajectory.
Velocity Penalty in the preplanning	Make FDDP preplanned trajectory more smooth and less aggressive.
Number of 500Hz offset steps	Simulate time delay caused by the solver's computation time, causing the solver input state and the actual system state to differ. Offsets are compensated by performing a forward integration using Pinocchio.
Noise on joint position observations	Use a random unit vector scaled by the noise factor to disturb the joint Position observations given by MuJoCo. Noise is filtered out by applying a filter on the Finite Difference computation for the joint velocities.
Noise on Optitrack observations	Use a random unit vector scaled by the noise factor to disturb each Optitrack observation vector. Noise is filtered out by performing a linear regression and an additional filter on the Finite Differences computation of the pendulum velocities.
Maximum Number of Iterations for the FDDP solver	Limit the internal maximum number of iterations to make the solver more robust against noise and state uncertainties.

TABLE III: Hyperparameters introduced in the MPC controller and their effect on the solver and the solution.

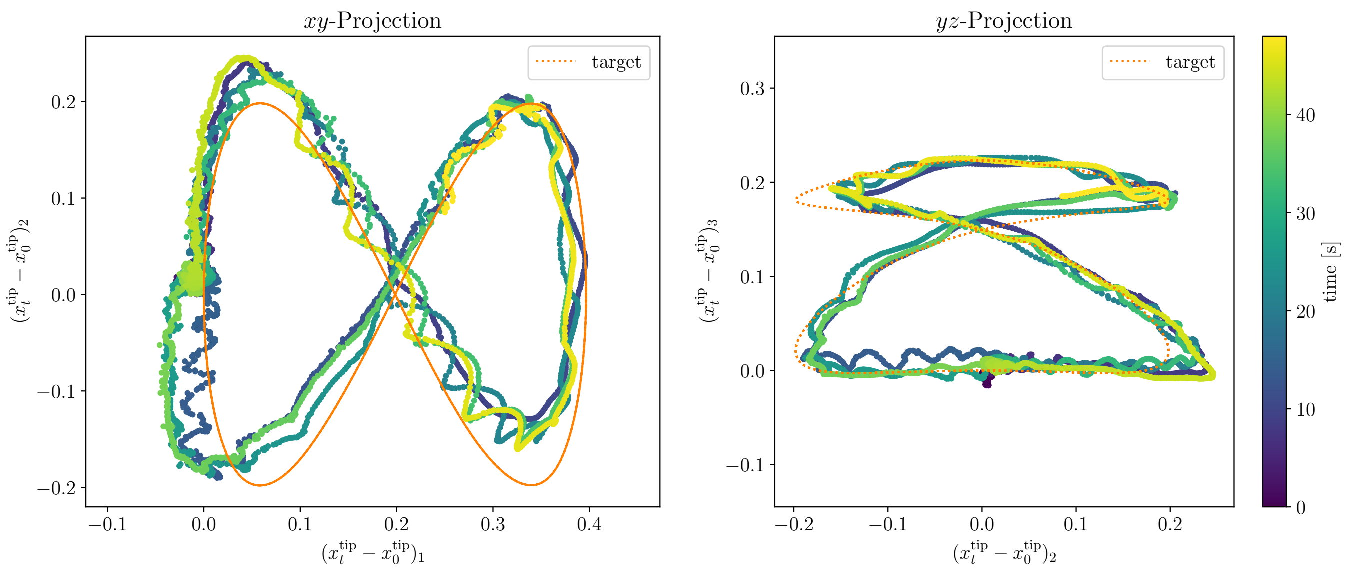


Fig. 11: Projected trajectories of the pendulum's tip position for the trajectory following task on the real system, solved with the LQR controller. The controller operates at 120 Hz and obtains an average tracking error of $3.2270 \cdot 10^{-2}$ m. The corresponding videos are available under [here].

and Automation (ICRA). 2021, pp. 10114–10120. DOI: 10.1109/ICRA48506.2021.9561093.

[11] Adam Heins and Angela P. Schoellig. *Keep it Upright: Model Predictive Control for Nonprehensile Object Transportation with Obstacle Avoidance on a Mobile Manipulator*. 2023. arXiv: 2305.17484 [cs.RO].

[12] Justin Carpentier et al. "The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives".

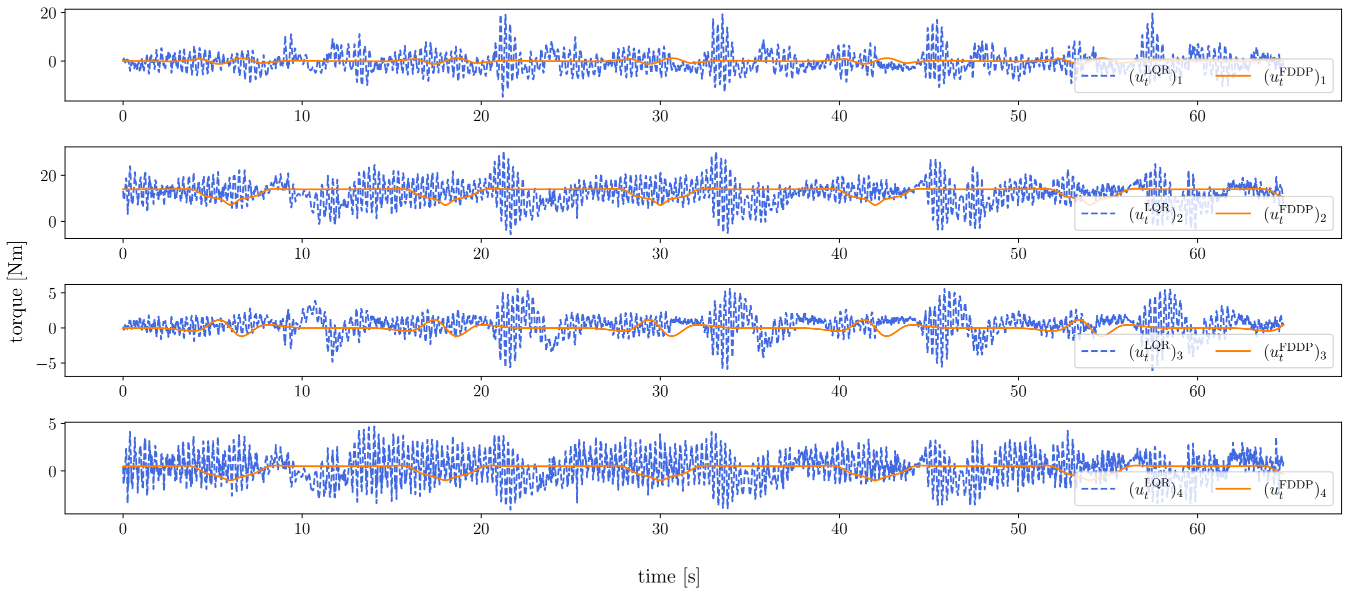


Fig. 12: Torques of the linear feedforward controller with PD-feedback (LQR in the plot) for the tractory tracking task (in simulation). The controller operates at $120\,\mathrm{Hz}$ and obtains an average tracking error of $3.2270 \cdot 10^{-2}\,\mathrm{m}$. The corresponding videos are available under [here].

MPC Penalty	No Noise (IP 1)	Noise, Delay & Friction (IP 2)
Control u	$1 \cdot 10^{-2}$	$1 \cdot 10^{-1}$
Tip position x^{tip}	$7 \cdot 10^{4}$	$6 \cdot 10^{5}$
Rotational angle	$1 \cdot 10^{5}$	$1 \cdot 10^{6}$
Resting position	$5 \cdot 10^{-2}$	$1 \cdot 10^{2}$
Joint velocities	$3 \cdot 10^{-2}$	1.0

TABLE IV: MPC solver configurations for the stabilization experiment without noise (IP 1) as well as with noise, delay and friction (IP 2).

In: *SII 2019 - International Symposium on System Integrations*. Paris, France, Jan. 2019. URL: `https://hal.laas.fr/hal-01866228`.

[13] C. Mastalli et al. “Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[14] C Mastalli et al. “A feasibility-driven approach to control-limited DDP”. In: *Autonomous Robots* 46.8 (2022), pp. 985–1005.

[15] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: `10.1109/IROS.2012.6386109`.

[16] Peng Song, Peter Kraus, and Pierre Dupont. “Analysis of Rigid-Body Dynamic Models for Simulation of Systems With Frictional Contacts”. In: *Journal of Applied Mechanics* 68 (Aug. 1999). DOI: `10.1115/1.1331060`.

[17] Jan Lunze. *Regelungstechnik 2 - Mehrgrößensysteme, Digitale Regelung*. Berlin Heidelberg New York: Springer-Verlag, 2010. ISBN: 978-3-642-10198-4.

Penalty	LQR (**Simulation**, Stabilization & Tracking)	LQR (**Real System**, Stabilization)	LQR (**Real System**, Tracking)
Control u	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$3 \cdot 10^{-2}$
Tip position x^{tip}	$1 \cdot 10^{3}$	$1 \cdot 10^{3}$	$1 \cdot 10^{3}$
Rotational angle	0.0	0.0	0.0
Resting position	$1 \cdot 10^{-1}$	$1 \cdot 10^{-1}$	5
Joint velocities	0.0	0.0	0.0

TABLE V: Solver configurations for the Linear Feedforward Controller with PD-feedback in simulation and on the real system. In order to avoid system crashes by decreasing the order of magnitude for the feedback matrices, we the increased the action penalty and the penalty on the setpoint rest.