Pendulum Acrobatics

(Integrated Project 1)

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 Barret WAM robotic arm, allowing for more challenging motions to be generated while simultaneously introducing a kinematic subtask in the control problem. We benchmark different MPC control schemes on both simple setpoint reaches as well as the generation of circular and spiral trajectories. The best performing method, that we implement on top of the Crocoddyl library, delivers convincing simulated results on all investigated trajectories. Supplementary material is available under [here].**

I. Introduction

First mentioned by Widrow and Smith [1] in 1964 and rediscovered by Barto, Sutton and Anderson in 1984 [2], 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.

With various real-world applications like biped locomotion for humanoid robots [3], rockets [4] and segways [5] in mind, over time researchers invented a large family of inverted pendulum systems with all kinds of different configurations, e.g. the double inverted pendulum, the 3D linear pendulum [6] or the spatial inverted pendulum [7, 8].

In the literature one can usually find three distinct application cases (cf. [8]): *swing-up control*, i.e. swing up the pendulum from a stable position (pendulum facing downwards) to an unstable position (pendulum facing upwards), *stabilization control*, in which the controller stabilizes the pendulum at an unstable position and *tracking control* where the directly actuated part (i.e. the cart) follows a reference trajectory while balancing the pendulum in an unstable position.

Until today, the inverted pendulum enjoys great popularity and is frequently used to test state-of-the-art learning methods like PILCO [9], Guided Policy Search [10] and appears as a benchmark in more recent actor-critic related Reinforcement Learning papers as Proximal Policy Optimization [11] or Trust Region Policy Optimization [12]. Additionally, the standard inverted pendulum is one of the predefined environments in Open AI's Gymnasium (former Open AI Gym) [13].

Our main contribution is to provide a tracking control framework for a three-dimensional spherical inverted pendulum mounted on the Barrett WAM arm[1] (cf. section I-B). We model the 3D spherical pendulum with two unactuated two-dimensional rotational joints with orthogonal rotational axes. Thus, it is a more challenging task as the pendulum can fall off in lateral direction too. The dynamics are complex and nonlinear.[2] The only related work we have seen so far are the approaches by Chung et al. in 1999 [15] to balance an three-dimensional inverted pendulum with a kinematically redundant planar-direct-drive robot in the xy-hyperplane and the paper by Albouy and Praly in 2000 [16] in which they present swing ups of a 3D spherical inverted pendulum on a simplified arm-like robot. Consequently, to the best of our knowledge, no one has ever worked on tracking control for a 3D spherical inverted pendulum. We extend this approach even further by working on tracking control based on the pendulum's tip position instead of the actuator's position in contrast to methods presented in the literature so far.

A. Aim of the project

In this first part of the Integrated Project we work in a simulation environment. The main goal is to explore how hard the proposed problem is and whether non-learning optimal control algorithms can provide solutions while being executable in real-time. Additionally, with part two of the Integrated Project in mind, we want to implement a framework to later on a) generalize the results to the real robot and b) act as an interface to execute movements generated by more advanced learning algorithms my supervisors work on. The aforementioned problem 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.

*Wintersemester 2022/23, Compiled on August 14, 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

[1] See `https://advanced.barrett.com/wam-arm-1`.

[2] Cf. [8] for a mathematical formulation of the dynamical system in a similar setting. In this project we do not model the system dynamics explicitly, instead we use the `.urdf`-description of the robot and the Pinocchio [14] library.

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. 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 reference positions and the pendulum's angles $(\theta_t)_{t\in[0,T]}$ with the z-axis are close to zero. I.e. we want to solve

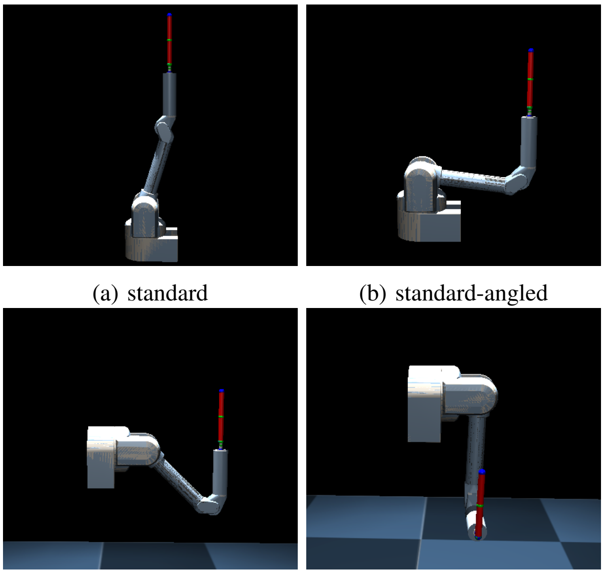
$$\begin{aligned}\min_{\substack{(\mathbf{q}_t)_{t\in[0,T]},\\(\mathbf{u}_t)_{t\in[0,T]}}} \quad & \int_0^T \mu_1(\mathbf{x}_t^{\text{tip}} - \mathbf{p}_t, t)\,\mathrm{d}t + \int_0^T \mu_2(\theta_t, t)\,\mathrm{d}t \\ \text{s.t.} \quad & \forall t \in [0,T]: \ \mathbf{u}_t \in [\mathbf{u}_{\min}, \mathbf{u}_{\max}], \\ & \mathbf{u}_t = M(\mathbf{q}_t)\ddot{\mathbf{q}}_t + c(\mathbf{q}_t, \dot{\mathbf{q}}_t) + g(\mathbf{q}_t), \\ & \mathbf{x}_t = f_{\mathrm{F}}(\mathbf{q}_t),\end{aligned} \tag{1}$$

for time-dependent weighting functions $\mu_1 : \mathbb{R}^3 \times [0,T] \to \mathbb{R}_{\geq 0}$, $\mu_2 : \mathbb{R} \times [0,T] \to \mathbb{R}_{\geq 0}$ (e.g. $\mu_1 \equiv \frac{1}{T}\|\cdot\|_2, \mu_2 \equiv \frac{1}{T}|\cdot|$ for the average error). 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}.

B. Experimental Setup

As we want to verify whether certain moves and trajectories require different starting configurations respectively if our experiments are setup independent, we work with two different setups of the Barret WAM (base plate facing upwards and base plate rotated by 90°). In order to compare the two different setups, we use two different configurations each, resulting in overall four configurations for the first experiment, see fig. 1. Additionally, with the Integrated Project 2 in mind, we want test the different configurations and determine the best one, which we will deploy on the real system. In every configuration, we use the (rotated) standard Barret WAM with a $30\,\mathrm{cm}$ pendulum that is connected to the robot through a small base plate of $5\,\mathrm{mm}$ height and a connection pole of $5\,\mathrm{cm}$ height.

Internally, we use the Pinocchio [14] based optimal control library Crocoddyl[3] [17] to plan trajectories and compute torques. To separate the solution and execution processes, like on the real system, we use the MuJoCo [18] simulation environment to execute the computed torques.[4] 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.



(a) standard

(b) standard-angled

(c) rotated

(d) human-like

Fig. 1: Visualizations of the four different configurations we used in our first experiment. For the corresponding initial joint positions see table I.

Configuration	Initial joint position q
standard	$(0, -0.3, 0, 0.3, 0, 0)^{\mathsf{T}}$
standard-angled	$(0, -1.7, 0, 1.7, 0, 0)^{\mathsf{T}}$
rotated	$(0, -0.78, 0, 2.37, 0, 0)^{\mathsf{T}}$
human-like	$(0, -1.6, 1.55, 1.6, 0, 1.55)^{\mathsf{T}}$

TABLE I: Initial joint positions q for our four configurations. We have 4 degrees of freedom and model the pendulum with two additional joints with orthogonal rotational axes (cf. section I).

II. Theoretical Introduction

In this section we motivate the formulation of problem (1) in a more general optimal control setting, explain different solution ideas and motivate the usage of a Differential Dynamic Programming (DDP) based Model Predictive Control (MPC) controller. Additionally, we emphasize how Crocoddyl's FDDP solver extends the classical DDP solver to effectively consider box constrained optimal control problems.

A. Problem formulation

Optimal control problems are a very powerful approach to couple motions as well as controls of a dynamical system by using task goals (e.g. costs or optimality conditions) with constraints (e.g. control bounds and system dynamics). Our problem in Definition 1 already has the natural structure of an optimal control problem. Thus, for the reasons of practicability, we only need to reformulate our original definition as a time-discrete optimal control problem.

For $N \in \mathbb{N}$ time steps $k = 0, \dots, N-1$, we consider the following generic discrete-time dynamical system

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k), \tag{2}$$

[3] See `https://github.com/loco-3d/crocoddyl`.

[4] This procedure requires a lot of fine-tuning to match the two different model descriptions and thanks to Kai we provide a script to test `.urdf` and `.xml` descriptions against each other.

where $\mathbf{x}_k = (\mathbf{q}, \mathbf{v}) \in X \subseteq \mathbb{R}^{n_x}$ is the state of the system at time step k lying a differential manifold formed by the joint configurations and its tangent vector. $\mathbf{u}_k \in \mathbb{R}^{n_u}$ is the control input or input torque at time step k and f represents the system dynamics given the current state and control.

Assume we are given a function $\ell : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ representing the running costs and a function $\ell_N : \mathbb{R}^{n_x} \to \mathbb{R}$ serving as the terminal cost model. Our goal is to find a sequence of states $\mathbf{X}^* := \{\mathbf{x}_0^*, \dots, \mathbf{x}_N^*\}$ and controls $\mathbf{U}^* := \{\mathbf{u}_0^*, \dots, \mathbf{u}_{N-1}^*\}$ which minimize the objective function

$$\begin{aligned} \mathbf{X}^*, \mathbf{U}^* = \underset{\mathbf{X} \subset \mathcal{X}, \mathbf{U} \subset \mathcal{U}}{\arg\min} \sum_{k=0}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k) + \ell_N(\mathbf{x}_N) \\ \text{s.t.} \quad \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \; \forall k \in \{0, \dots, N-1\} \end{aligned} \tag{3}$$

for sets of admissable solutions $\mathcal{X}, \mathcal{U}$ which encode e.g. box constraints on the control and the state of the system. Each variable $\mathbf{x}_k \in \mathbf{X}^*, k = 0, \dots, N-1$ is called *decision variable*.

Optimal control problems of the form of (3) can be solved by using *direct methods* which apply a discretization on the state and the control and solve (large) systems of linear equations to satisfy the First-Order-Necessary-Conditions (FONC), i.e. the Karush-Kuhn-Tucker (KKT) conditions.[5] Although this approach benefits from applying the current state-of-the-art Nonlinear Programming (NLP) solvers, approaches of this type typically suffer under complex trade-offs between feasibility and optimality which usually have to be handled by manually defining merit functions (cf. [20, p. 435 ff.]). Futhermore, they do not encode the temporal or Markovian structure of the problem given by eq. (2) and thus usually require huge matrix factorizations, i.e. making them rather slow.

To encode the Markovian property and exploit the problem's natural sparse structure, rollout based *multiple shooting methods* have arisen and seem promising. By using Bellman's principle of optimality, rollout based methods can reduce the complexity of the problem and thus only need comparably small matrix factorizations. Consequently, methods of this type are more likely applicable in a Model Predictive Control (MPC) setting, allowing the control law to react to perturbations as well as to overcome the problem of only having guaranteed feasibility along the nominal trajectory.

By defining the value function

$$V_k(\mathbf{x}) := \min_{\mathbf{U}} \sum_{i=k}^{N-1} \ell(\mathbf{x}_i, \mathbf{u}_i) + \ell_N(\mathbf{x}_N)$$

at time step k as the optimal cost-to-go function given a starting points $\mathbf{x}$, we can rewrite the optimal value function by using the recursive nature of Bellman's principle of optimality via

$$V_k(\mathbf{x}) = \min_{\mathbf{u}} \ell(\mathbf{x}, \mathbf{u}) + V_{k+1}(f(\mathbf{x}, \mathbf{u})) \tag{4}$$

and with

$$V_N(\mathbf{x}) = \ell_N(\mathbf{x}) \tag{5}$$

as the boundary condition.

B. Differential Dynamic Programming

Differential Dynamic Programming (DDP) is an iterative algorithm to numerically solve nonlinear optimal control problems of the aforementioned structure. As DDP was already captured in the Robot Learning lecture, we will only provide a brief overview and instead focus on the improvements proposed by Mastalli et al. in [17, 21] compared to the classical DDP solver [22]. For a detailed derivation see section VIII-B. Note that we temporarily omit the box constraints on the control for simplicity.[6]

The main idea of DDP is to improve the current estimate of the state-control trajectory $(\mathbf{X}, \mathbf{U})$, i.e. the (current) nominal trajectory, by using a local linear quadratic (QP) approximations of the system dynamics and the loss function along the nominal trajectory and iteratively performing a *backward pass* and a *forward pass (rollout)*, i.e. solve a Sequential Quadratic Program (SQP).

In order to solve problem (3), we need the following assumption.

Assumption 1: From now, on we assume that there is a shooting node (i.e. a decision variable) for each integration step along the trajectory.

C. Feasibility driven DDP (FDDP)

This section will emphasize the key differences between the classical DDP algorithm and the feasibility driven DDP (FDDP) algorithm proposed by Mastalli et al. Furthermore, we will highlight implementation details of the Crocoddyl library.

1) Limitations of DDP: One of the biggest challenges in DDP algorithms is the handling of (in)equality constraints and multiple papers[7] try solve this issue with various different approaches.

As already mentioned in the derivation of the DDP algorithm (cf. section VIII-B), another issue is the need for a feasible warmstart. Since the dynamics functions f is chained multiple times during the rollout, the algorithm is highly sensitive to the initial control choice and finding a non-trivial warmstart[8] is a challenging task.

[5] For more information about Nonlinear Programming, FOCNs and KKT conditions we refer to the German standard reference of Ulbrich and Ulbrich [19] or the English equivalence of Nocedal and Wright [20].

[6] We will see in Section II-C.1 that most of the classical solvers fail to handle control constraints efficiently.

[7] Cf. [23] for DDP with equality constraints and [24, 25] for DDP with inequality constraints.

[8] Obtaining a state trajectory $\mathbf{X}_0$ that serves as an initial guess for the optimal control solver is straight-forward. However, determining a corresponding control sequence $\mathbf{U}_0$ beyond quasi-static maneuvers is in practice fairly difficult and inconvenient. 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.

2) FDDP: Mastalli et al. promise with their FDDP solver and the corresponding implementation in the Crocoddyl library to not only efficiently handle box constraints for the control but also improve solutions for tasks involving highly nonlinear dynamics (e.g. flips, jumps). As our project involves an instable system with highly nonlinear dynamics, we focus on the usage of Crocoddyl in this first part of the Integrated Project.

According to Mastalli et al., both issues mentioned in Section II-C.1 can be narrowed down to

$$\bar{\mathbf{f}}_k = f(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_{k+1} = 0,$$

i.e. the necessity of closed gaps (cf. eq. (16)). Gaps in the dynamics lead to derivatives in deviating decision variables and consequently to poor convergence results. Furthermore, the coercion to close the gaps limits the solver's possibilities as the search space is drastically narrowed down. The main idea of Mastalli et al. to overcome the problem of closed gaps, is to allow infeasible trajectories in the backward pass of the algorithm by introducing five modifications of the (classical) DDP algorithm. A brief summary of the key differences between the classical DDP algorithm and the FDDP method is shown in table II. For a more detailed comparison, please consider section VIII-B.3.

Overall, the FDDP solver promises to be a powerful improved DDP algorithm with the possibility to handle highly nonlinear dynamics, accept infeasible warmstarts and still provide the same rate of convergence as the classical DDP algorithm.

3) Theoretical Limitations: Although numerous experiments[9] demonstrate the impressive capabilities of FDDP, currently there are no theoretical guarantees that FDDP is able to find a solution. As already mentioned above, based on only changing the gradients in the Ricatti recursion and not the Hessian matrices, the authors claim a rate of superlinear convergence for the FDDP solver. Despite this rate of convergence is proved for the classical DDP algorithm and FDDP seems to be able to confirm this conjecture experimentally, this claim seems rather optimistic as even the existence of a solution is up to this point unclear.

4) Implementation details of Crocoddyl: Crocoddyl provides two specific implementation details which make the library really attractive from a user's perspective. First, Crocoddyl internally uses analytic derivatives (compared to finite differences or algorithm differentiation, cf. [31, 32, 33]) which naturally allows to exploit the aforementioned sparse structure of the problem and ensures impressively fast computations. Second, Crocoddyl uses a strict separation of models and data which avoids dynamic memory allocation. Additionally, the library does not only provide a low-level C++ implementation, but also thinly wrapped Python bindings to allow fast prototyping which is really important for our project.

Crocoddyl internally satisfies Assumption 1 by enforcing the user to define decision variable respectively *cost nodes* first and then integrating the nodes along the given trajectory. Consequently, this enables the user to customize each node individually and results in a lot of flexibility.

D. Model Predictive Control

As already discussed in the Robot Learning lecture and in the previous sections, all DDP-based algorithm 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 an Model Predictive Control (MPC) control law. Since we are working with an instable system with highly nonlinear dynamics, the choice of an FDDP based MPC controller seems justified.

III. Experiment 1: Setpoint reaches

In order to get a feeling of how difficult our goal is in general, we designed a benchmark test 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 1D-hyperplane. We use the 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 $x^{\text{tip}}_{t=0} + (-0.15, 0, 0)^{\mathsf{T}}$. Consequently, enumerating all the different combinations yields 18 different tasks in total.

A. Task Space MPC without preplanning

To obtain an initial benchmark, we start with the naive idea of formulating the setpoint reach as a raw task space problem. As already mentioned in section I-B, one of our goals is to possibly identify one model that performs best which we can afterwards deploy on the real robot in the IAS laboratory.

1) Experiment configuration: We experimented with different MPC planning horizons but due to computational limitations imposed by the complexity of the problem, we were not able to run an example with more than 20 time steps in reasonable time. Thus, we use three different MPC horizons of $\{5, 10, 20\}$ time steps to study how the horizon affects the solution. Clearly, we expect better results for higher planning horizons at the cost of higher computational time.

The MuJoCo simulation is by default running with a control frequency of $500\,\mathrm{Hz}$ yielding a integration constant of $\Delta_t^{\mathrm{MJ}} = 0.002\,\mathrm{s}$ when discretizing problem (1) over time. Having the real system in mind, running at $120\,\mathrm{Hz}$, we use three different factors ($\text{factor} \in \{1, 2, 4\}$) to increase our internal integration time, yielding the following three cases for an exemplary MPC horizon of 10 time steps:

1 We use $\mathrm{d}t = 1 \cdot \Delta_t^{\mathrm{MJ}} = 0.002\,\mathrm{s}$ with a real-time control frequency of $500\,\mathrm{Hz}$ and real-time MPC planning horizon of $0.02\,\mathrm{s}$.

[9] Cf. [26, 27] for applications of Crocoddyl in an optimal control setting and [28, 29, 30] for MPC applications.

Difference	Improvement
Decouple dynamics and control feasibility	Expand solution space and simplify theoretical formulation
Include gaps in the Ricatti equations	Allow infeasible guesses in backward pass
Nonlinear rollout	Enable infeasible guesses in the forward pass
Goldstein linesearch method	Ensure contraction rate and avoid ascent directions
Regularization of Hessians	Stabilize switch between steepest descent and Newton step

TABLE II: Key differences between the FDDP solver and the classical DDP algorithm.

2 We use $\mathrm{d}t = 2 \cdot \Delta_t^{\mathrm{MJ}} = 0.004\,\mathrm{s}$ with a real-time control frequency of $250\,\mathrm{Hz}$ and real-time MPC planning horizon of $0.04\,\mathrm{s}$.

4 We use $\mathrm{d}t = 4 \cdot \Delta_t^{\mathrm{MJ}} = 0.008\,\mathrm{s}$ with a real-time control frequency of $125\,\mathrm{Hz}$ and real-time MPC planning horizon of $0.08\,\mathrm{s}$.

Thus, we can investigate the tradeoff between an increased MPC planning horizon and a decreased real-time control frequency, which allows us to verify whether the results degenerate with lower intervention points.

To formulate our problem in Crocoddyl, we use penalty terms on the position of the pendulum's tip, the rotation of the pendulum as well as a small regularization term on the torques. Since we observed a strong oscillation behavior of the elbow in our first experiments, we introduce an additional regularization penalty on the configuration of the robot in the state space with respect to its initial configuration. All of the penalty terms are tuned manually and individually for each configuration but kept fix for each configuration to allow comparability between the different experiments.

Of course, here is room for improvement as we cannot ensure that we have found the best penalty terms as well as one can manually tune individual combinations to achieve better results.

For the setpoint benchmark test, we use a time horizon of 3000 nodes, i.e. 3000 time steps in the MuJoCo simulation, resulting in a time horizon $T = 3000 \cdot \Delta_t^{\mathrm{MJ}} = 6\,\mathrm{s}$. To evaluate whether a specific configuration satisfactory reached the setpoint, we consider the last 10% respectively 300 nodes and measure the average absolute positional error in meters and the average absolute angular error in degrees.

2) Results: We provide tables with the average absolute rotational[10] and absolute positional error over the last 300 nodes, i.e. the last $0.6\,\mathrm{s}$ of the trajectory for all of the different configurations as well as the corresponding penalty configurations and the computational times under [here].[11]

For the penalty parameters the most interesting point is that for all configurations we always need a higher penalty on the rotational error than on the positional error. The fine tuning of the penalty parameters in general is insanely complex as the system and thus the solver is enormously sensitive with respect to the chosen parameters. For the reason of stability, one should always impose more focus on the rotational error than on the positional error, as the solver is in general not able to recover from situations where the pendulum deviates too strongly from the balanced position, almost always resulting in a crash of the system.

In general, the rotational errors are impressively low as we strongly focus on keeping the pendulum balanced. Thus, an average rotational error larger than 10^{+1} is an indicator for an aforementioned crash of the system and a collapse of the simulation. Furthermore, one can observe by manually visualizing movements for different MPC horizons, that especially in the $500\,\mathrm{Hz}$ cases, controllers with a lower planning horizon show strong oscillation behavior. This seems reasonable, as the controller is not able to see far enough into the future but realizes that oscillating allows reactions in nearly all directions at any time. Nevertheless, this obviously results in non-optimal movements. Additionally, we can also see that in general each configuration seems to have a favorable direction which is particularly simple to achieve (e.g. standard-angled in z-direction).

If we compare the different configurations, we can observe that although for the highest real-time control frequency the standard-angled configuration works the best, with an increase in the integration time, the rotated and human-like configurations perform better and interestingly both fairly similar. The standard configuration performs in nearly all cases rather poorly since the initial joint positions are really close to a singularity which clearly limits the controller's possibilities.

The most important and really interesting part is the evolution of the results with respect to the increase of the integration factor. While we can clearly see that the computational time drastically decreases and a factor of 4 almost already allows real-time applictions, we do not observe a degeneration of the results. This shows that the tradeoff between a longer planning horizon and a lower real-time control frequency is favorable for the longer planning horizon as for instance the results of the rotated configuration even improve with an increase of the integration factor.

As already mentioned in the introduction of this chapter, our expectation of higher MPC horizons yielding better results proves to be true, although for example the 10-rotated configuration seems to be a tiny bit better than the 20-rotated configuration, which is an unexpected result.

In conclusion, our results already look really promising and we are especially satisfied with the impressive results regarding the rotational error. Nevertheless, we have still a lot of room for improvements. From a computational

[10] We compute the angle in degree via the standard cosine-formula between the z-axis and the pendulum.

[11] This corresponds to using the weighting functions $\mu_1(x,t) := \|x\|_2$, $\mu_2(x,t) := |x|$ for the solver and $\bar{\mu}_1(x,t) := \mathbb{1}_{[t_{\mathrm{eval}},T]}(t) \cdot 1/(T - t_{\mathrm{eval}}) \cdot \|x\|_2$, $\bar{\mu}_2(x,t) := \mathbb{1}_{[t_{\mathrm{eval}},T]}(t) \cdot 1/(T - t_{\mathrm{eval}}) \cdot |x|$ with $T = 6\,\mathrm{s}$ and $t_{\mathrm{eval}} = 5.4\,\mathrm{s}$ for the evaluation (cf. definition 1).

perspective, we can of course gain major speedups by using Crocoddyl's C++ implementation instead of the Python bindings and our pipeline of solving and executing is still serial and not yet parallelized. Moreover, we have chosen the parameters manually and we kept them fixed for each configuration across different integration times. From a control engineering perspective this is not a good practice, as technically each of the experiments yields a totally different system which has to be treated individually. We will work on this part in detail during the Integrated Project 2, when we switch to the real robot. Most importantly, the trajectories we obtain by our controller are in general smooth but not yet physically optimal. Currently, our MPC controller seems to have problems with the aforementioned 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 fig. 2.

B. Improve the results by following a precomputed trajectory

Based on the results presented in the previous section as well as having the real system in mind, we will from now on focus on the rotated configuration with a factor of 4 for the integration time, i.e. a real-time control frequency of $120\,\mathrm{Hz}$ and an integration time step of $\mathrm{d}t = 0.008\,\mathrm{s}$.

Compared to the previous approach, we will now include an additional preplanning step in order to balance the aforementioned tradeoff between the positional and the rotational error and thus allow our controller to follow more complex trajectories. When using the FDDP solver for our setpoint problem and visualizing the result (under the assumption of a perfect model), we obtain trajectories similar to the results shown in fig. 2. Hence, it seems promising to 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.

This naive approach immediately yields three problems:

1) By default, 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. Consequently, we need to simplify the precomputed solutions.
2) 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. Simply applying an additional (quadratic) penalty on the rotation of the pendulum eliminates the benefits of the precomputed optimal movements as the MPC is again not able to allow tipping movements.
3) The precomputed trajectories consistently exhibit a sinking behavior when the controller is close to the global end of the time horizon. This can (most likely) be explained by the fact that in the end it is worth it to slightly increase the positional penalties and instead eliminate the costs on the torques by slowly dropping the end effector.

We address these issues individually by using the approaches presented below.

Smoothing the trajectories by applying velocity penalties in the joint space: Additional penalties on the joint space velocity during the FDDP preplanning allow us to smooth the precomputed trajectories as shown in fig. 3. Besides a decreased magnitude of the velocity, we also observe that the smoothed trajectory shows a less aggressive tipping movement, i.e. the angles are smaller and the displacement of the end effector is also reduced. We tried different values for the velocity penalty but experimentally chose $v_{\mathrm{pen}} = 0.5$.[12]

Follow the trajectories in the task space: An elegant solution to overcome problem 2 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. Our experiments show that this gives more freedom to the MPC controller. Additionally, inspired by gain scheduling in control engineering (cf. [34, p. 223 ff.]), we use a clipped quadratic barrier penalty function to allow tipping movements similar to the preplanned movement. In particular, the preplanned trajectory also contains information about the angle of the pendulum at each time step and we only penalize angles which are larger than the precomputed ones.[13]

Cutting off the precomputed trajectories: To solve issue 3, we simply cut off the precomputed task space trajectory after 1500 steps, i.e. $3\,\mathrm{s}$, and mirror the last point. This procedure is equivalent to increasing the planning horizon of the FDDP solver in the precomputation and then cutting off, as the FDDP solver always perfectly matches the desired setpoint. Practically, any time step between 1500 and 2800 would be possible.

1) Results: As before, the results are available [here]. We use the same evaluation strategy as in section III-A.2. Additionally, we provide three videos comparing the results of the original MPC, the precomputed FDDP solution and the improved MPC controller which are available [here].[14]

In total, the results of our improved method are really impressive. Figure 4 exemplarily shows the results for the $(x, 0.1, -1)$ experiment and we can clearly see that the preplanned MPC controller (PMPC) outperforms the MPC of section III-A and nearly perfectly matches the trajectory computed by Crocoddyl's FDDP solver. As expected, the MPC controller with the biggest planning horizon of 20 time

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

[13] This is implemented by computing the corresponding rotation vector at each time step and applying a pointwise clipped quadratic loss function on the current pendulum's orientation.

[14] The slight rotating maneuver in the end of the third video can be explained by the fact that the controller tries to minimize the required torques to stabilize the setpoint. This is observable by plotting the torques in the course of time.

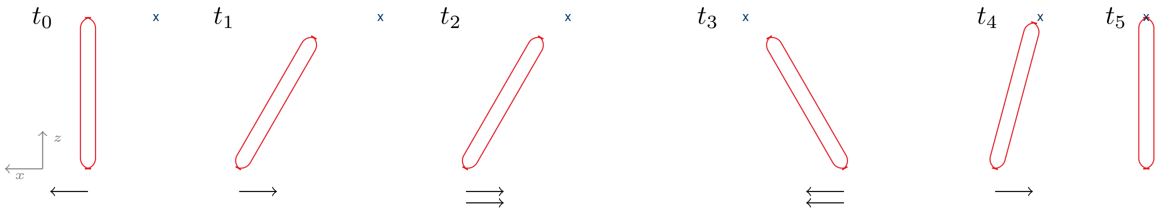


Fig. 2: Optimal setpoint movement with bang-bang control for the target $(x, 0.1, -1)$ and schematic time steps $t_1, \ldots t_5$. The target in the task space is marked by the blue cross and the torques are sketched by arrows (two arrows representing a stronger force). The y-axis points out of the image plane. In reality the steps shown at time steps t_2 and t_3 are repeated with lower and lower torques (as sketched in t_4) resulting in a diminishing oscillating behavior around the setpoint (see e.g. fig. 3).

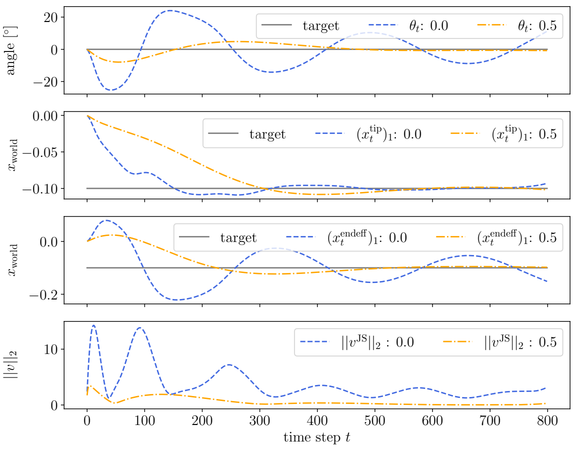


Fig. 3: Visualization of the velocity penalty's impact on the precomputed FDDP trajectory for the $(x, 0.1, -1)$ experiment. In blue we can see the FDDP solution without an additional velocity penalty and the orange curves visualize the precomputed solution with an additional quadratic penalty on the joint space velocity scaled with $v_{\text{pen}} = 0.5$. In contrast to the tabular based evaluation, 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. This allows negative angles and emphasizes the movement proposed in fig. 2. Note that only the first 800 steps, i.e. 1.6 s, of the trajectories are shown.

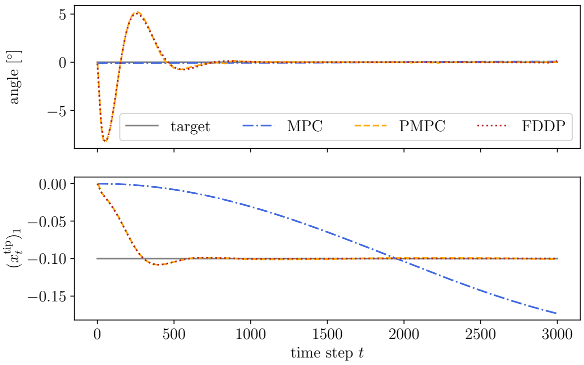
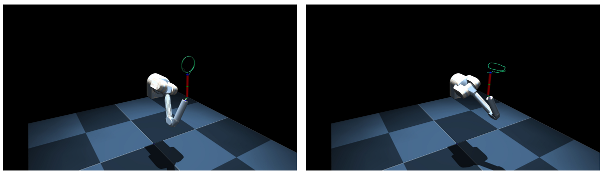


Fig. 4: Comparison of the results for the MPC controller presented in section III-A, the precomputed FDDP trajectory and the resulting trajectory of the preplanned MPC (PMPC) controller, all for the setpoint experiment $(x, 0.1, -1)$. As in fig. 4, we do not plot the absolute rotational error but instead 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 III in the appendix.

steps achieves the best results. The positional errors being in the order of magnitude 10^{-4} mean that we are in the millimeter range which is a major improvement compared to our previous results. As already mentioned, we scaled the angle barrier with a scalar β to verify whether the MPC can still follow the trajectory with less freedom, i.e. $\beta < 1$, or achieve even better results with less restrictive bounds, i.e. $\beta > 1$. The former did not yield stable solutions and the latter shows no improvements, so keeping $\beta = 1$ is justified. 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 also lower compared to the first experiment and since they are measured in degree, we can see that the controller is able to perfectly balance the pendulum. The computational time is adequate and we achieve real time applicability in nearly all cases. Although the computational time of the FDDP solver does not matter, we can nevertheless observe that the solver is impressively fast. Most interestingly, shifting the tradeoff between positional and rotational error into the precomputation, allows us to use a higher positional penalty and the resulting ratio between the rotational and positional penalty is closer to one in comparison to the first experiment. To summarize, we consider the setpoint reach task to be solved.

IV. Experiment 2: Trajectory following

In our final experiment, we try to generalize our results to time-varying trajectories: a circular movement, being embedded in the two-dimensional xz-hyperplane, and a spiral



(a) Circle

(b) Spiral

Fig. 5: Last frame for the circular respectively the spiral movement. The parametrization in section VIII-C is added to the starting position of the pendulum's tip at time step $t = 0$ and we follow the trajectory with the tip of the pendulum. All penalty terms are exactly the same as in the second experiment but we added an additional penalty on the joint space velocity. The coloring of the trajectory is inversely proportional to $(\|\dot{\mathbf{x}}_t\|)_{t\in[0,T]}$. For the circular respectively the spiral movement we have computational times of $6.89\,\mathrm{s}$ and $6.87\,\mathrm{s}$ as well as average absolute positional errors of $6.32 \cdot 10^{-3}\,\mathrm{m}$ and $6.77 \cdot 10^{-3}\,\mathrm{m}$ over 3000 nodes, i.e. $6\,\mathrm{s}$, in total. This corresponds to using the weighting functions $\mu_1(x,t) := \|x\|_2$, $\mu_2(x,t) := |x|$ for the solver and $\bar{\mu}_1(x,t) := 1/T \cdot \|x\|_2$, $\bar{\mu}_2(x,t) := 1/T \cdot |x|$ for the evaluation (cf. definition 1). A comprehensive comparison of the results is presented in table IV in the appendix.

movement, representing a real three-dimensional movement. Both of the trajectories' parametrizations are derived in section VIII-C and we define the task space target by adding the proposed parametrization onto the pendulum's tip position at time step $t = 0$.

Except for adding a small penalty parameter on the joint velocities and removing the velocity penalties for the FDDP preplanning, all parameters are exactly the same as in the previous experiment.

Figure 5 exemplarily shows two images with the pendulum's tip position in the last frame of the simulation and a colored trajectory of the previously visited positions in the task space. Videos for both of the trajectories can be found [here]. We tried different variations of frequencies, radii and heights which all worked equally good and we encourage you to checkout our code under [here] to experiment with the trajectories by yourself. Due to time and space restrictions we cannot provide a full analysis as for the previous two experiments.

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

V. Evaluation of the Library

In summary, Mastalli et al. provide with their FDDP solver in Crocoddyl a really impressive and enormously fast algorithm. Apart from the experiments mentioned in section II-C.3, we can confirm that FDDP is able to find optimal solutions at an astonishing computation speed. Furthermore, the algorithm really surprised us with its numerical stability when we experimented with (log-)barrier functions (cf. section VIII-D). Here, one normally has to increase the hyperparameter β but Crocoddyl is able to provide optimal solutions even when directly calling the solver with large values of β.

Nevertheless, Crocoddyl still has issues with non-perfect warmstarts although the quasi-static maneuvers described in section II-C.1 partly solve that issue. Additionally, Crocoddyl only provides a limited set of cost and activation functions and only allows activation functions with a diagonal Hessian matrix. Furthermore, due to the structure of the (F)DDP algorithm, one cannot combine cost nodes over multiple time steps and thus e.g. minimum jerk trajectories are not possible. In our experiments we could not confirm that FDDP is able to find solutions the (classical) DDP solver in Crocoddyl is not able to find. However, in all tests the FDDP solver proved to be significantly faster.

VI. Software Contribution

Our software is publicly available under [here] and licensed under the [CC BY 4.0] license. We provide a thin wrapper around the raw Crocoddyl bindings including several factory methods to allow an easy to use and quick testing framework for arbitrary trajectories with individually defined cost nodes. Our code structure is similar to Crocoddyl and thus fully modularized. Furthermore, we provide a memory and time efficient Python implementation of an MPC controller including (optional) FDDP-based warmstarts. Additionally, we implemented C++ code and the corresponding Python bindings of the log barrier functions for the Crocoddyl versions 1.8.1 and 1.9.0 (cf. section VIII-D). Since Crocoddyl internally works with soft joint limits and efficiently works with our log barrier implementation, this could be interesting to anyone suffering under the necessity of non-violable joint limits.

Thanks to Kai we provide MuJoCo helper function, e.g. to validate `.urdf` and `.xml` files and visualize task space positions of MuJoCo sites over time.[15]

VII. Outlook and current work

Our next major goal is to transfer our results onto the real robot in the IAS laboratory. To achieve this, we plan to transfer our code into a C++ version to obtain computational speedups. Since we cannot longer rely on the simulation sensors to return the pendulum's position and orientation, we plan to use *OptiTrack*[16] as a substitution. One problem we expect on the real system is the construction of a pendulum joint which has so little friction that we can enable elbow moves presented in the videos.

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 and your creative ideas and endless motivation to experiment with new solution ideas. Thank you very much!

[15] See `https://github.com/kploeger/python_sandbox`.

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

VIII. APPENDIX

A. Full Results of the shown Experiments

The comprehensive results of the examples shown in fig. 4 and fig. 5 are presented in table III respectively table IV. All results are available under [here].

B. DDP Derivation

We follow the notation of [25], [22] and [21].

1) Backward pass: We start by reformulating our problem (4). For a state-action pair $(\mathbf{x}_k, \mathbf{u}_k)$, we define the cost-to-go function of taking a given action in a given state (i.e. the action value function) $Q : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ in terms of the deviation (δ_x, δ_u) from the nominal state-action pair $(\mathbf{x}_k, \mathbf{u}_k)$, i.e.

$$\begin{aligned} Q_k(\delta_x, \delta_u) &:= \ell(\mathbf{x}_k + \delta_x, \mathbf{u}_k + \delta_u) \\ &\quad + V_{k+1}(f(\mathbf{x}_k + \delta_x, \mathbf{u}_k + \delta_u)). \end{aligned} \tag{6}$$

Next, we Taylor-expand of Q_k around $(\mathbf{0}, \mathbf{0})$ and drop terms with order higher than two, yielding

$$\begin{aligned} Q_k(\delta_x, \delta_u) &= Q_k(\mathbf{0}, \mathbf{0}) + Q_{x,k}^\top \delta_x + Q_{u,k}^\top \delta_u \\ &\quad + \frac{1}{2}\left({\delta_x}^\top Q_{xx,k} \delta_x + {\delta_u}^\top Q_{uu,k} \delta_u\right) \\ &\quad + {\delta_u}^\top Q_{ux,k} \delta_x. \end{aligned} \tag{7}$$

With A_{k+1} being the (symmetric) Hessian of V_{k+1} evaluated at $\mathbf{x}_{k+1}$ and b_{k+1} the gradient respectively (we will determine these quantities soon via a Ricatti recursion update).[17] We conclude

$$\begin{aligned} Q_k(\mathbf{0}, \mathbf{0}) &= \ell(\mathbf{x}_k, \mathbf{u}_k) + V_{k+1}(\mathbf{x}_{k+1}) \\ Q_{x,k} &= \ell_{x,k} + f_x^\top b_{k+1} \\ Q_{u,k} &= \ell_{u,k} + f_u^\top b_{k+1} \\ Q_{xx,k} &= \ell_{xx,k} + f_x^\top A_{k+1} f_x + b_{k+1}^\top f_{xx} \\ Q_{uu,k} &= \ell_{uu,k} + f_u^\top A_{k+1} f_u + b_{k+1}^\top f_{uu} \\ Q_{ux,k} &= \ell_{ux,k} + f_u^\top A_{k+1} f_x + b_{k+1}^\top f_{ux} \end{aligned}$$

with the abbreviations $\ell_{u,k} := \partial_u \ell(\mathbf{x}_k)$, etc. As we already did in the lecture, we can now express the value function in terms of a minimization problem of Q over δ_u by

$$V_k(\mathbf{x}) = \min_{\delta_u} Q_k(\delta_x, \delta_u). \tag{8}$$

Since we already obtained the quadratic approximation of Q_k, the analytics solution to (8) can be easily computed by setting the derivative to zero

$$0 \overset{!}{=} \partial_{\delta_u} Q_k(\delta_x, \delta_u) = Q_{u,k} + Q_{uu,k}\delta_u + Q_{ux,k}\delta_x, \tag{9}$$

yielding the solution

$$\delta_u := -Q_{uu,k}^{-1}\left(Q_{ux,k}\delta_x + Q_{u,k}\right). \tag{10}$$

Thus, between δ_x and δ_u we have the relation

$$\delta_u = K_k \delta_x + k_k, \tag{11}$$

with $K_k = -Q_{uu,k}^{-1} Q_{ux,k}$ and $k_k = -Q_{uu,k}^{-1} Q_{u,k}$ for the optimal linear feedback controller.

Next, we can focus on how to efficiently obtain the quantities A_{k+1} and b_{k+1} by performing a Ricatti recursion update. We plug the optimal control (11) into the approximated Q function (7) to recover an approximation of the value function

$$V_k(\mathbf{x}) = \frac{1}{2}\left({\delta_x}^\top A_k \delta_x + b_k^\top \delta_x + c_k\right), \tag{12}$$

with an (irrelevant) constant c_k and components

$$\begin{aligned} A_k &= Q_{xx,k} + K_k^\top Q_{uu,k} K_k + Q_{ux,k}^\top K_k + K_k^\top Q_{ux,k}, \\ b_k &= Q_{x,k} + K_k^\top Q_{uu,k} k_k + Q_{ux,k}^\top k_k + K_k^\top Q_{u,k}. \end{aligned} \tag{13}$$

Consequently, by starting with $A_N = \ell_{xx}^N$ and $b_N = \ell_x^N$, we can recursively compute the optimal feedback control in each decision variable by solving K_k, k_k, A_k and b_k backwards in time for $k = N-1, \dots, 0$.

In conclusion, we see that Bellman's principle of optimality lays the foundation for the backward pass, since it allows us to break the optimal control problem into smaller subproblems which can then be solved analytically in a time-reversed recursive manner. It is noteworthy that we need a feasible nominal trajectory, either by a previous forward-pass or a given warmstart, otherwise we cannot perform a backward pass. In practice this is a big inconvenience and thus addressed by Mastalli et al.

2) Forward pass (Rollout): In the second step of the algorithm, we update the nominal trajectory by applying the optimal linear feedback (11) from the previous backward pass. By starting with the initial boundary condition $\mathbf{x}_0^{\text{new}} = \mathbf{x}_0$ we obtain:

$$\mathbf{u}_k^{\text{new}} = \mathbf{u}_k + \delta_u = \mathbf{u}_k + K_k(\mathbf{x}_k^{\text{new}} - \mathbf{x}_k) + k_k, \tag{14}$$

$$\mathbf{x}_{k+1}^{\text{new}} = f(\mathbf{x}_k^{\text{new}}, \mathbf{u}_k^{\text{new}}). \tag{15}$$

Note that (15) can be seen as explicitly closing the gaps between all decision variables, i.e.

$$\bar{\mathbf{f}}_k = f(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_{k+1} = 0 \tag{16}$$

for $k = 0, \dots, N-1$.

3) Feasibility DDP: There are mainly five modifications proposed by Mastalli et al. to overcome the problems mentioned in section II-C.1.

Decoupling dynamic constraints and feasibility: FDDP internally focuses first on searching a (dynamically) feasible search direction (*feasibility-driven direction*) by opening respectively ignoring the control bounds and afterwards tries to satisfy the control constraints (*control-bound direction*). This opens up possibilities for the solver to find more and better trajectories as the search space is not immediately narrowed down by the control constraints. Additionally, it is an open research question how one can quantify the effects of gaps on the control bounds. Thus, it is currently not possible to optimize for dynamical and control feasibility at the same time. Mastalli et al. conjecture and show experimentally that this separation can enable FDDP to find movements which

[17] We need the Hessian and the gradient of the value function evaluated in the decision variable $\mathbf{x}_{k+1}$, as we consider the value function evaluated at $f(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_{k+1}$, cf. eq. (2).

	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 III: Results for the setpoint experiment $(x, 0.1, -1)$ shown in fig. 4.

	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 IV: Results for the time-varying trajectory experiment shown in fig. 5.

would be not computable by DDP due to violated control bounds in the first iterations.

Allow infeasible dynamics: By including the gaps in the gradient of the Ricatti equations (cf. eq. (13)), Mastalli et al. enable dynamically infeasibility (i.e. open gaps) during the backward pass. Thus, the solver does not suffer under the necessity of a feasible warmstart. Since only the gradient in the Ricatti equations changed, the authors conjecture that the rate of convergence in the limit is superlinear.[18]

Nonlinear rollout: In order to allow infeasible guesses in the backward pass, we need to adjust the forward pass accordingly. By introducing a step size α, we change the classical forward pass defined in (14) and (15) to:

$$\mathbf{x}_0^{\text{new}} = \mathbf{x}_0 - (1-\alpha)\bar{\mathbf{f}}_0, \tag{17}$$

$$\mathbf{u}_k^{\text{new}} = \mathbf{u}_k + K_k(\mathbf{x}_k^{\text{new}} - \mathbf{x}_k) + \alpha k_k, \tag{18}$$

$$\mathbf{x}_{k+1}^{\text{new}} = f(\mathbf{x}_k^{\text{new}}, \mathbf{u}_k^{\text{new}}) - (1-\alpha)\bar{\mathbf{f}}_k, \tag{19}$$

for $k = 0, \ldots, N-1$. Both, the updated forward pass and the contraction rate of $(1-\alpha)$, arise from deriving the KKT conditions of the modified optimal control problem, i.e. the equality constraints of the system's dynamics now also include the gaps. Additionally, the nonlinear rollout allows the algorithm to check for search directions more accurately. Note that the classical DDP algorithm always closes the gaps, i.e. $\alpha = 1$.

Linesearch method: Internally, FDDP uses the Goldstein [20, p. 36] linesearch method instead of the Armijo [35] rule for two reasons. First, due to the aforementioned modifications of the algorithm, FDDP cannot ensure anymore that each search direction is a descent direction. By using the Goldstein rule, the authors avoid using ascent directions. Second, as shown in equation (19), the step size α gives a contraction rate of $(1-\alpha)$ for the gaps. Thus, the choice of the stepsize is crucial for the algorithm and the usage of a more advanced linesearch method seems reasonable.

Regularization: From an optimization perspective a rather classical but nevertheless powerful modification is the usage of regularization schemes on the Hessian matrices of the Q function during the Ricatti recursion. According to Mastalli et al. regularization significantly increases the stability of the algorithm when it switches between using Steepest Descent and Newton's direction and vice versa.

In conclusion, the FDDP solver promises to be a powerful improved DDP algorithm with the possibility to handle highly nonlinear dynamics, accept infeasible warmstarts and still provide the same rate of convergence as the classical DDP algorithm.

C. Trajectory Parametrization

We give the following parametrizations for the circular movement (embedded in the xz-hyperplane)

$$\begin{aligned} \varphi_{\text{C}} : [0, 2\pi) &\to \mathbb{R}^3 \\ \theta &\mapsto r \cdot (\sin(\omega\theta), 0, \cos(\omega\theta))^{\mathsf{T}}, \end{aligned}$$

with an angular velocity $\omega \in \mathbb{Z}^{\times}$ which also allows to encode a direction (e.g. $\omega < 0$) and a radius $r > 0$. Embeddings into different hyperplanes can be onbtained permuting the coordinates in the definition of $\text{im}(\varphi)$ accordingly. The spiral movement is given by

$$\begin{aligned} \varphi_{\text{S}} : [0, 2\pi) &\to \mathbb{R}^3 \\ \theta &\mapsto (r \cdot \sin(\omega\theta), r \cdot \cos(\omega\theta), h \cdot \theta)^{\mathsf{T}}, \end{aligned}$$

with radius $r > 0$, height scaling $h > 0$ and an angular velocity $\omega \in \mathbb{Z}^{\times}$.

D. Barrier Penalty Function

Currently, in Crocoddyl only penalty functions with a clipping mechanism that applies a penalty when leaving the set of feasible points (e.g. encoded via box constraints) are provided. This leads in general to two problems:

1) We cannot ensure hard costraints. For instance even constraints on the joint limits are only soft-constraints and can technically be violated by the solver, if the structure of the cost functions favors a violation. For practical applications this is extremely dangerous and inconvenient at the same time, as we need to impose and tune additional penalty parameters for the joint limits.
2) As in our case with the pendulum, when dealing with highly instable systems, a penalty term only occurring *after* the bounds are already violated can be too late for the system and the solver to recover. Thus, we need slowly increasing penalty terms which yield larger

[18] Intuitively this makes sense, as the rate of convergence in the limit is determined by the Hessian matrices in the Ricatti equations which stay unchanged.

penalties as we approach the boundary of the feasible set.

One classical NLP solution approach is to use so called *barrier methods* or *interior point methods* which shift inequality constraints into the cost function. By following Ulbrich and Ulbrich [19, p. 136 ff.] respectively [20, p. 583 ff.], we encode real-valued inequalities of the form $x \leq 0$ with a so-called *(log-)barrier function* of the form

$$b : (-\infty, 0) \to \mathbb{R}, x \mapsto b(x) = -\ln(-x).$$

One can generalize the barrier function to encode vector-values inequalities of the form $A\mathbf{x} \leq \mathbf{b}$ for vectors $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ and a matrix $A \in \mathbb{R}^{n \times n}$.

In our case, we are a) only interested in component-based box constraints for the angles $(\theta_t)_t$ and b) Crocoddyl only supports Hessian matrices with a diagonal structure. Thus, we we only need a component-wise weighting vector $\mathbf{w} \in \mathbb{R}^n$ (in order to e.g. turn off penalties for specific components) and we introduce an additional scalar $\beta > 0$ to obtain

$$\phi_\beta : \mathbb{R}^n \to \mathbb{R} \tag{20}$$

$$\mathbf{x} \mapsto -\frac{1}{\beta} \sum_{i=1}^{n} \ln(\mathbf{b}_i - \mathbf{w}_i \mathbf{x}_i), \tag{21}$$

with component-wise upper bounds $\mathbf{b} \in \mathbb{R}^n$.

The severity of the barrier is controlled by the choice of the *barrier parameter* β, where small β leads to a gradual barrier and large β to a sharp barrier. Here, one usually has to perform a tradeoff, as larger values of β are required to obtain points close to the border of the feasible set but simultaneously cause steep gradients as well as ill-conditioning and thus make the problem harder to solve. Consequently, in practice we usually generate a strictly monotonically increasing sequence $(\beta_n)_{n\in\mathbb{N}}$ and solve the problem multiple times, each time using the previous solution as a warmstart [19, p. 136]. Figure 6 shows an exemplary barrier function for varying values of β.

Note that the curvature of the barrier function significantly increases for larger values of β, which drastically slows down Newton's method in practical applications and is the reason why primal-dual-interior-point methods emerged.

In order to add our function to Crocoddyl, we need to compute the gradient and the Hessian matrix. First, we compute the gradient w.r.t. $\mathbf{x}$. For a component $i \in \{1, \dots, n\}$ we have

$$\partial_i \phi_\beta(\mathbf{x}) = -\frac{1}{\beta} \frac{1}{\mathbf{b}_i - \mathbf{w}_i \mathbf{x}_i} \cdot (-\mathbf{w}_i) = \frac{1}{\beta} \frac{\mathbf{w}_i}{\mathbf{b}_i - \mathbf{w}_i \mathbf{x}_i}. \tag{22}$$

By using **component-wise operations**, we can rewrite the gradient as

$$\nabla_x \phi_\beta(\mathbf{x}) = \frac{1}{\beta} \frac{\mathbf{w}}{\mathbf{b} - \mathbf{w} \odot \mathbf{x}}.$$

Similarly, we can compute the entry (i, i), for $i \in \{1, \dots, n\}$ of the Hessian matrix as

$$\left(\nabla^2_{xx} \phi_\beta(\mathbf{x})\right)_{ii} = \frac{1}{\beta} \frac{\mathbf{w}_i^2}{(\mathbf{b}_i - \mathbf{w}_i \mathbf{x}_i)^2},$$

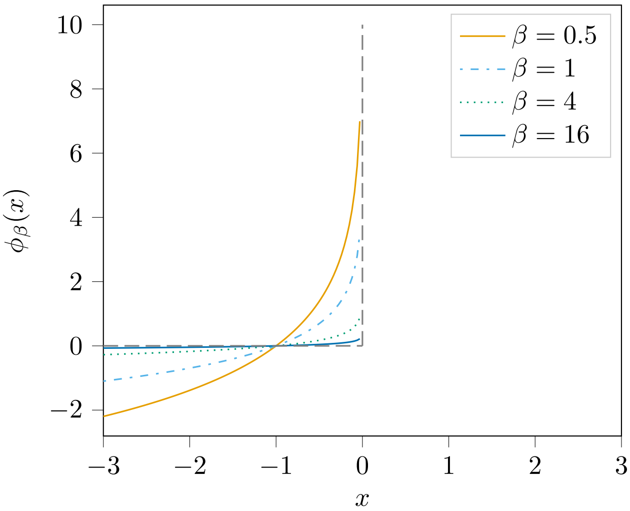


Fig. 6: Barrier function for different values of β and an upper bound of $b = 0$.

again, in closed form

$$\nabla^2_{xx} \phi_\beta(\mathbf{x}) = \frac{1}{\beta} \text{diag}\left(\frac{\mathbf{w}^2}{(\mathbf{b} - \mathbf{w} \odot \mathbf{x})^2}\right).$$

Our implementation of the gradient respectively the Hessian in our Crocoddyl-fork is based on the aforementioned closed forms for faster vectorized computations. Our implementation is tested with version 1.9.0 and 1.8.1 of Crocoddyl.[19]

In practical applications, one can easily model (one-dimensional) box constraints of the form $x \in [x_{\min}, x_{\max}] \subset \mathbb{R}$ by using a barrier function of the form

$$\phi_\beta(x) = \frac{1}{\beta} \left(\ln(x_{\max} - x) + \ln(-x_{\min} + x)\right).$$

In general, for n-dimensional component-wise box-constraints one always obtains $2n$ terms in the sum. In the case of symmetric boc-constraints one can simplify the equation above even further by applying log-rules and the binomial formula.

Our own experiments show the impressive numerical stability and sensitivity of the Crocoddyl solver. As mentioned before, one usually has to push $\beta \nearrow \infty$ and use the previous iterates as a warmstart in order to model sharp feasibility boundaries and obtain good results. In contrary, although the computation time is slows down, we were able to achieve convincing results by directly using large values of β.

It is noteworthy that despite barrier functions face the downside of enforcing a feasible starting point, i.e. in our case the pendulum has to be within the angular constraints. Otherwise the solver is due to the structure of the barrier penalty function not able to perform a backward pass. For us, this does not constitute a disadvantage, but in general it is a disadvantage when working with barrier functions.

[19] See [Crocoddyl 1.9.0] and [Crocoddyl 1.8.1].

References

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

[2] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (1983), pp. 834–846. DOI: 10.1109/TSMC.1983.6313077.

[3] Ahmed Elhasairi and Alexandre Pechev. "Humanoid Robot Balance Control Using the Spherical Inverted Pendulum Mode". In: *Frontiers in Robotics and AI* 2 (2015). ISSN: 2296-9144. DOI: 10.3389/frobt.2015.00021. URL: https://www.frontiersin.org/articles/10.3389/frobt.2015.00021.

[4] C.W. Anderson. "Learning to control an inverted pendulum using neural networks". In: *IEEE Control Systems Magazine* 9.3 (1989), pp. 31–37. DOI: 10.1109/37.24809.

[5] Wael Younis and Mohamed Abdelati. "Design and Implementation of an Experimental Segway Model". In: 1107 (Mar. 2009). DOI: 10.1063/1.3106501.

[6] S. Kajita et al. "The 3D linear inverted pendulum mode: a simple modeling for a biped walking pattern generation". 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, 239–246 vol.1. DOI: 10.1109/IROS.2001.973365.

[7] 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.

[8] 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.

[9] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 465–472. ISBN: 9781450306195.

[10] Sergey Levine and Vladlen Koltun. "Guided Policy Search". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1–9. URL: https://proceedings.mlr.press/v28/levine13.html.

[11] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[12] John Schulman et al. "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477 (2015). arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.

[13] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[14] Justin Carpentier et al. "The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives". In: *SII 2019 - International Symposium on System Integrations*. Paris, France, Jan. 2019. URL: https://hal.laas.fr/hal-01866228.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Michael Ulbrich and Stefan Ulbrich. *Nonlinear optimization. (Nichtlineare Optimierung.)* German. Basel: Birkhäuser, 2012. ISBN: 978-3-0346-0142-9/pbk.

[20] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006.

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

[22] Rohan Budhiraja et al. "Differential Dynamic Programming for Multi-Phase Rigid Contact Dynamics". In: *IEEE-RAS International Conference on Humanoid Robots (ICHR)*. 2018.

[23] Daniel M. Murray and Sidney J. Yakowitz. "Constrained differential dynamic programming and its application to multireservoir control". In: *Water Resources Research* 15.5 (1979), pp. 1017–

1027. DOI: https://doi.org/10.1029/WR015i005p01017. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/WR015i005p01017. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/WR015i005p01017.

[24] Yuichiro Aoyama et al. *Constrained Differential Dynamic Programming Revisited*. 2020. DOI: 10.48550/ARXIV.2005.00985. URL: https://arxiv.org/abs/2005.00985.

[25] Zhaoming Xie, C. Karen Liu, and Kris Hauser. "Differential dynamic programming with nonlinear constraints". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 695–702. DOI: 10.1109/ICRA.2017.7989086.

[26] Rohan Budhiraja, Justin Carpentier, and Nicolas Mansard. "Dynamics Consensus between Centroidal and Whole-Body Models for Locomotion of Legged Robots". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 6727–6733. DOI: 10.1109/ICRA.2019.8793878.

[27] Josep Marti-Saumell et al. "Squash-Box Feasibility Driven Differential Dynamic Programming". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 7637–7644. DOI: 10.1109/IROS45743.2020.9340883.

[28] Ewen Dantec et al. "Whole Body Model Predictive Control with a Memory of Motion: Experiments on a Torque-Controlled Talos". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 8202–8208. DOI: 10.1109/ICRA48506.2021.9560742.

[29] Thomas Corbères et al. "Comparison of predictive controllers for locomotion and balance recovery of quadruped robots". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 5021–5027. DOI: 10.1109/ICRA48506.2021.9560976.

[30] Sebastien Kleff et al. "High-Frequency Nonlinear Model Predictive Control of a Manipulator". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 7330–7336. DOI: 10.1109/ICRA48506.2021.9560990.

[31] Joel A E Andersson et al. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

[32] Arvind U. Raghunathan, Devesh K. Jha, and Diego Romeres. "PYROBOCOP : Python-based Robotic Control & Optimization Package for Manipulation and Collision Avoidance". In: *CoRR* abs/2106.03220 (2021). arXiv: 2106.03220. URL: https://arxiv.org/abs/2106.03220.

[33] Sergio Lucia et al. "Rapid development of modular and sustainable nonlinear model predictive control solutions". In: *Control Engineering Practice* 60 (Mar. 2017), pp. 51–62. DOI: 10.1016/j.conengprac.2016.12.009.

[34] Jürgen Adamy. *Nichtlineare Systeme und Regelungen -*. Berlin Heidelberg New York: Springer-Verlag, 2014. ISBN: 978-3-642-45013-6.

[35] Larry Armijo. "Minimization of functions having Lipschitz continuous first partial derivatives." In: *Pacific Journal of Mathematics* 16.1 (1966), pp. 1–3. DOI: pjm/1102995080. URL: https://doi.org/.