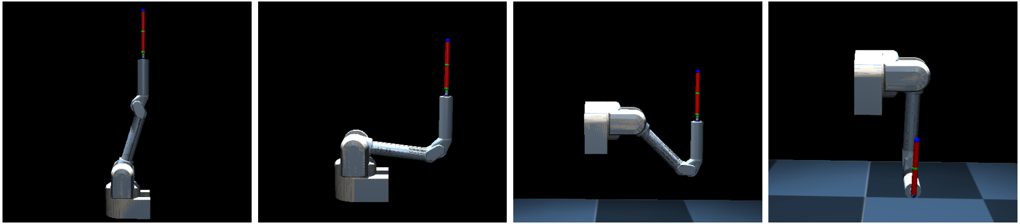
Integrated Project 1: Pendulum Acrobatics

Florian Wolf, supervised by: Pascal Klink and Kai Ploeger

Experimental Setup and Aim of the Project
Four different configurations

- Three-dimensional spherical cart pole system at end-effector of Barrett WAM robotic arm
- Complex system:
  - Highly nonlinear, instable and underactuated
  - Requires fast-reactive controller
  - Can fall off in lateral direction too



(a) standard (b) standard-angled (c) **rotated** (d) human-like

Experimental Setup and Aim of the Project

- Three-dimensional spherical cart pole system at end-effector of Barrett WAM robotic arm
- Complex system:
  - Highly nonlinear, instable and underactuated
  - Requires fast-reactive controller
  - Can fall off in lateral direction too
- Four different configurations tested in total
- Generic Goal (**Tracking Control**):

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

Formulation as an Optimal Control Problem

Given:

- Time horizon $T \in \mathbb{R}_{>0}$, Sequence of reference points $(\mathbf{p}_t)_{t\in[0,T]} \subset \mathbb{R}^3$ (Task Space)
- Time-dependent weighting functions $\mu_1 : \mathbb{R}^3 \times [0,T] \to \mathbb{R}_{\geq 0}$, $\mu_2 : \mathbb{R} \times [0,T] \to \mathbb{R}_{\geq 0}$

Goal: Solve

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

$\Rightarrow$ OC Library Crocoddyl (Mastalli et al. 2020) with Feasibility driven DDP (FDDP) solver in MPC setting

Setpoint Reaches, Experiment 1: Task Space MPC controller

Problem description

- Experience difficulty of the problem for all four configurations
- Reach setpoints (direction, distance, orientation)
  - Direction: $\{x, y, z\}$
  - Distance: $\{0.05\,\text{m}, 0.1\,\text{m}, 0.15\,\text{m}\}$
  - Orientation: $\{-1, 1\}$

  e.g. $(x, 0.15, -1)$ corresponds to reaching: $x_{t=0}^{\text{tip}} + (-0.15, 0, 0)^\intercal$
- Control frequencies and integration time steps (MuJoCo and Crocoddyl)
  - $500\,\text{Hz} \rightsquigarrow \Delta_t^{\text{MJ}} = 0.002\,\text{s}$
  - $250\,\text{Hz} \rightsquigarrow \Delta_t^{\text{MJ}} = 0.004\,\text{s}$
  - $125\,\text{Hz} \rightsquigarrow \Delta_t^{\text{MJ}} = 0.008\,\text{s}$
- MPC horizons of $5, 10, 20$ time steps

  $\Rightarrow$ Trade-off between lower real-time control frequency and longer MPC horizon

$\Rightarrow$ In total 648 experiments, full data see [here]

Setpoint Reaches, Experiment 1: Task Space MPC controller

Evaluation Method and Results

- 3000 nodes (i.e. 6 s), average absolute error over the last 300 nodes (0.6 s)
- error space, rotation and computational time
- rotated configuration works best
- **Video 1:**
  - Positional error: $9.237 \cdot 10^{-2}$ [m]
  - Rotational error: $2.284 \cdot 10^{-1}$ [deg]
  - Computation time: $7.924 \cdot 10^{0}$ [s] (close to real time, still with Python Bindings)
- Physically non-optimal behavior (maybe some words w.r.t. smoothing)
- Problem: Trade-off between rotational and positional error

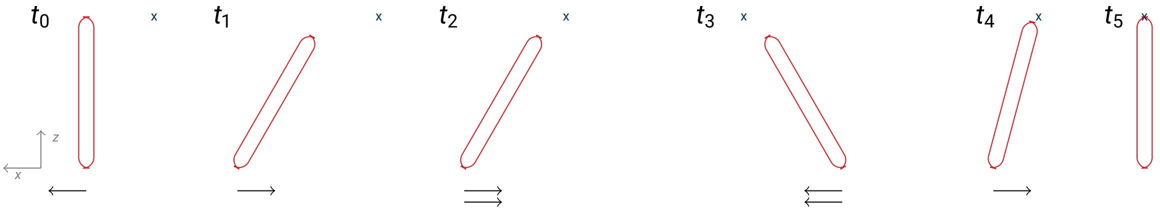


Figure: Optimal setpoint movement with bang-bang control for the target $(x, 0.1, -1)$ and schematic time steps $t_1, \ldots t_5$. Reality: steps t_2 and t_3 are repeated with diminishing oscillation behavior.

$\Rightarrow$ FDDP provides physically optimal trajectories

Setpoint Reaches, Experiment 1: Preplanned MPC

Use FDDP to precompute optimal trajectories

- Use FDDP solver to precompute optimal trajectory
  - Before: $(\mathbf{x}_t^{targ})_t := (\mathbf{x}_{t=0}^{\mathrm{tip}} + \mathrm{setpoint})_t$
  - Now: $(\mathbf{x}_t^{targ})_t := \mathrm{f}_{\mathrm{F}} \left(\mathrm{FDDP} \left((\mathbf{x}_{t=0}^{\mathrm{tip}} + \mathrm{setpoint})_t \right) \right)$ (forward kinematics of precomputed trajectory)

  $\Rightarrow$ solves issue with tradeoff between rotational and positional error
- Follow the precomputed trajectory in the task space (gives controller more freedom)
- Smooth precomputed trajectories with an additional velocity penalty (avoid Bang-Bang-Control)
- **Video 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]	$7.924 \cdot 10^{+00}$	$\mathbf{6.011 \cdot 10^{+00}}$	$9.704 \cdot 10^{-01}$

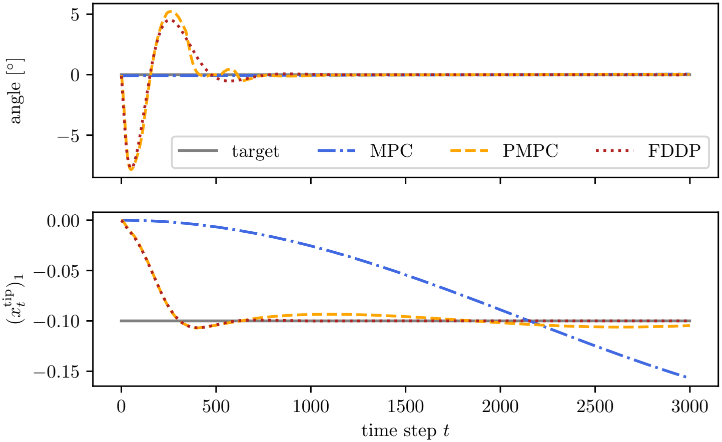


Figure: MPC controller vs. Preplanned solution (FDDP) vs. Preplanned MPC Controller for the setpoint experiment $(x, 0.1, -1)$. Plots for the y- and z-axis are not shown since the errors are on average below 10^{-3}.

$\Rightarrow$ Consider setpoint task to be solved.

Exeriment 2: Time-varying Trajectories with Preplanned MPC

- Two trajectories:
  - Circle: embedded in two-dimensional hyperplane
  - Spiral: full three-dimensional movement
- Use exactly the same setting and configuration as before, but we can:
  - Remove velocity penalty in the precomputation (trajectories have an inherent velocity)
- **Video 3:**

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

Conclusion and Outlook

Experimental results:

- Near perfect results for setpoint reaches
- Convincing results for trajectories in space
- Near real-time performance without parallelization and by just using Python Bindings

Programming Results:

- Crocoddyl with FDDP solver provides a really impressive solver
  - Optimal trajectories with incredibly fast performance
  - Python bindings
- Our code is available under [here]:
  - Thinly wrapped Python package with factory methods for trajectory-based cost models
  - Easy to use and suitable for fast prototyping

Next big goal: Transfer the results to the real robot (IP2)

Conclusion and Outlook

Experimental results:
- Near perfect results for setpoint reaches
- Convincing results for trajectories in space
- Near real-time performance without parallelization and by just using Python Bindings

Programming Results:
- Crocoddyl with FDDP solver provides a really impressive solver
  - Optimal trajectories with incredibly fast performance
  - Python bindings
- Our code is available under [here]:
  - Thinly wrapped Python package with factory methods for trajectory-based cost models
  - Easy to use and suitable for fast prototyping

Next big goal: Transfer the results to the real robot (IP2)

Thank you for your Attention!