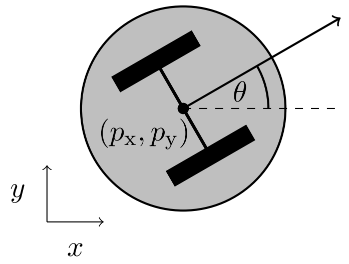
Summer School on Robust Model Predictive Control with CasADi
James B. Rawlings, Joel Andersson, Sergio Lucia, Moritz Diehl
University of Freiburg, September 15 to 19, 2025
`www.syscop.de/event/rmpc25`

Exercise 3: Robust Dynamic Optimization with `IPOPT`

Florian Messerer, Jonathan Frey, Katrin Baumgärtner, Moritz Heinlein

In this exercise, we implement and ellipsoidal tube OCP as well as a multistage OCP for a simple robot model. The aim is to drive a robot from its starting position p_{start} to a goal position p_{target} while robustly avoiding an obstacle.



The state of the robot is given by $x = (p_{\text{x}}, p_{\text{y}}, \theta, v, \omega)$, where $p_{\text{x}}, p_{\text{y}}$ parametrize the 2D-position of the center of the robot, θ is the heading angle, v the forward velocity and ω the angular velocity. The controls $u = (a, \alpha)$ are the forward acceleration a and angular acceleration α We assume that both the forward acceleration as well as the angular acceleration are subject to additive disturbances $w = (w_a, w_\alpha)$ yielding the ODE

$$\dot{p}_{\text{x}} = v\cos\theta, \tag{1a}$$
$$\dot{p}_{\text{y}} = v\sin\theta, \tag{1b}$$
$$\dot{\theta} = \omega, \tag{1c}$$
$$\dot{v} = a + \sigma_a w_a, \tag{1d}$$
$$\dot{\omega} = \alpha + \sigma_\alpha w_\alpha \tag{1e}$$

with σ_a, σ_α determining the scaling of the disturbances. This allows us to consider the unit sphere for the values of w_k, $w_k \in \mathcal{E}(0, I)$. We discretize the continuous-time system using one step of an explicit Runge-Kutta integrator of order 4 on an integration interval of length $h = 0.35$, with piecewise constant controls and disturbances to obtain the discrete-time dynamics

$$x_{k+1} = f(x_k, u_k, w_k). \tag{2}$$

As stage and terminal cost we use

$$l(x, u) = \gamma l_{\text{Huber}}(p - p_{\text{target}}) + u^\top R u, \tag{3a}$$
$$l_N(x) = \gamma_N l_{\text{Huber}}(p - p_{\text{target}}) + \tau v^2, \tag{3b}$$

with $p = (p_{\text{x}}, p_{\text{y}})$, weights R, γ, γ_N, τ, and where the pseudohuber loss $l_{\text{Huber}}(p) = \sqrt{p^\top p + 1}$ behaves like a quadratic function near the origin and like the unsquared 2-norm for larger values. This yields a smooth tracking behavior near the target position while proportionally penalizing larger distances.
The constraints include upper and lower bounds on the controls, bounds on the position, as well as an obstacle avoidance constraints,

$$u_{\min} \leq u \leq u_{\max}, \tag{4a}$$
$$p_{\min} \leq p, \tag{4b}$$
$$r_{\text{obs}} \leq \|c - p_{\text{obs}}\|_2, \tag{4c}$$

where r_{obs} denotes the radius of the obstacle and p_{obs} its center.

Nominal OCP

The nominal OCP, which we obtain by setting $w_k = 0$ for $k = 0, \dots, N-1$, takes the form

$$\begin{aligned}
\min_{\bar{x},\,\bar{u}} \quad & \sum_{k=0}^{N-1} l_k(\bar{x}_k, \bar{u}_k) + l_N(\bar{x}_N) && \text{(5a)}\\
\text{s.t.} \quad & \bar{x}_0 = \bar{\bar{x}}_0, && \text{(5b)}\\
& \bar{x}_{k+1} = f_k(\bar{x}_k, \bar{u}_k, 0), \quad k = 0, \dots, N-1, && \text{(5c)}\\
& 0 \geq h_k(\bar{x}_k, \bar{u}_k), \qquad k = 0, \dots, N-1, && \text{(5d)}\\
& 0 \geq h_N(\bar{x}_N), && \text{(5e)}
\end{aligned}$$

with $\bar{u} = (\bar{u}_0, \dots, \bar{u}_{N-1})$, $\bar{x} = (\bar{x}_0, \dots, \bar{x}_N)$.

Tasks

1. Run `main.py` in order to solve the nominal OCP (5). This yields us a baseline to which we can compare the robust OCP solutions obtained in the following.

Ellipsoidal-tube robust OCP

We consider the corresponding ellipsoidal-tube robust OCP in the form

$$
\begin{aligned}
\min_{\bar{x}, \bar{u}, P, \beta} \quad & \sum_{k=0}^{N-1} l_k(\bar{x}_k, \bar{u}_k) + l_N(\bar{x}_N) && && \text{(6a)}\\
\text{s.t.} \quad & \bar{x}_0 = \bar{\bar{x}}_0, && && \text{(6b)}\\
& P_0 = 0, && && \text{(6c)}\\
& \bar{x}_{k+1} = f_k(\bar{x}_k, \bar{u}_k, 0), && k = 0, \dots, N-1, && \text{(6d)}\\
& P_{k+1} = \psi_k(\bar{x}_k, \bar{u}_k, P_k), && k = 0, \dots, N-1, && \text{(6e)}\\
& 0 \geq h_k(\bar{x}_k, \bar{u}_k) + \sqrt{\beta_k + \varepsilon^2}, && k = 0, \dots, N-1, && \text{(6f)}\\
& \beta_k \geq H_k(\bar{x}_k, \bar{u}_k, P_k), && k = 0, \dots, N-1, && \text{(6g)}\\
& 0 \geq h_N(\bar{x}_N) + \sqrt{\beta_N + \varepsilon^2}, && && \text{(6h)}\\
& \beta_N \geq H_N(\bar{x}_N, P_N), && && \text{(6i)}\\
& 0 \leq \beta_k, && k = 0, \dots, N. && \text{(6j)}
\end{aligned}
$$

Here, $\bar{x}$, $\bar{u}$ are the nominal trajectories (the centers of the ellipsoids). The state ellipsoids are described by $P = (P_0, \dots, P_N)$, i.e., the state tube is $x_k \in \mathcal{E}(\bar{x}_k, P_k)$. The linearization-based ellipsoid dynamics are given by

$$\psi_k(\bar{x}_k, \bar{u}_k, P_k) := A_k(\bar{x}_k, \bar{u}_k) P_k A_k(\bar{x}_k, \bar{u}_k)^\top + \Gamma_k(\bar{x}_k, \bar{u}_k)\Gamma_k(\bar{x}_k, \bar{u}_k)^\top \tag{7}$$

with $A_k := \nabla_x f_k(\bar{x}_k, \bar{u}_k, 0)^\top$, $\quad B_k := \nabla_u f_k(\bar{x}_k, \bar{u}_k, 0)^\top$, $\quad \Gamma_k := \nabla_w f_k(\bar{x}_k, \bar{u}_k, 0)^\top$, $\quad k = 0, \dots, N-1$. We point out that P_k are symmetric matrices, such that for an implementation of the OCP it can make sense to only consider their lower or upper triangular part.
For the constraint backoffs we introduce the lifting variables β. This makes it easier to ensure that throughout the solver iterations we never encounter the square root of a negative number. The values of β correspond to the squared backoffs and are given via

$$H_k^i(\bar{x}_k, \bar{u}_k, P_k) = \nabla h_k^i(\bar{x}_k, \bar{u}_k)^\top P_k \nabla h_k^i(\bar{x}_k, \bar{u}_k), \tag{8}$$
$$H_N^i(\bar{x}_N, P_N) = \nabla h_N^i(\bar{x}_N)^\top P_N \nabla h_N^i(\bar{x}_N), \tag{9}$$

for $k = 0, \dots, N-1$ and $i = 1, \dots, n_{h_k}$, and with superscript ^{i} denoting the i-th component of the vector valued functions h_k and H_k.
Note: The vector valued functions H_k are defined componentwise. For the implementation in code, we recommend to follow this componentwise definition, with an explicit loop over index $i = 1, \dots, n_{h_k}$, instead of trying to avoid the loop via matrix-vector notation / operators.

Tasks

1. Complete the file `solver_open_loop_robust_ocp.py`, and modify and run `main.py` accordingly. Does the resulting trajectory differ from the nominal trajectory? In what way and why?

2. In its uncertainty prediction, the above robust OCP only considers an open loop control trajectory $\bar{u}$. This gives rise to unnecessarily large state tubes. Instead, we want to consider the linear feedback law $u_k = \bar{u}_k + K(x_k - \bar{x}_k)$, with feedback gain $K \in \mathbb{R}^{n_u \times n_x}$. For K, we use a heuristic structure, tailored to the robot model, given in the template as part of the OCP dataclass. Modify the OCP in order to incorporate this simple feedback law. How do you need to change the ellipsoid propagation and the constraint robustification? Keep in mind that the control trajectory is now also an uncertain variable.

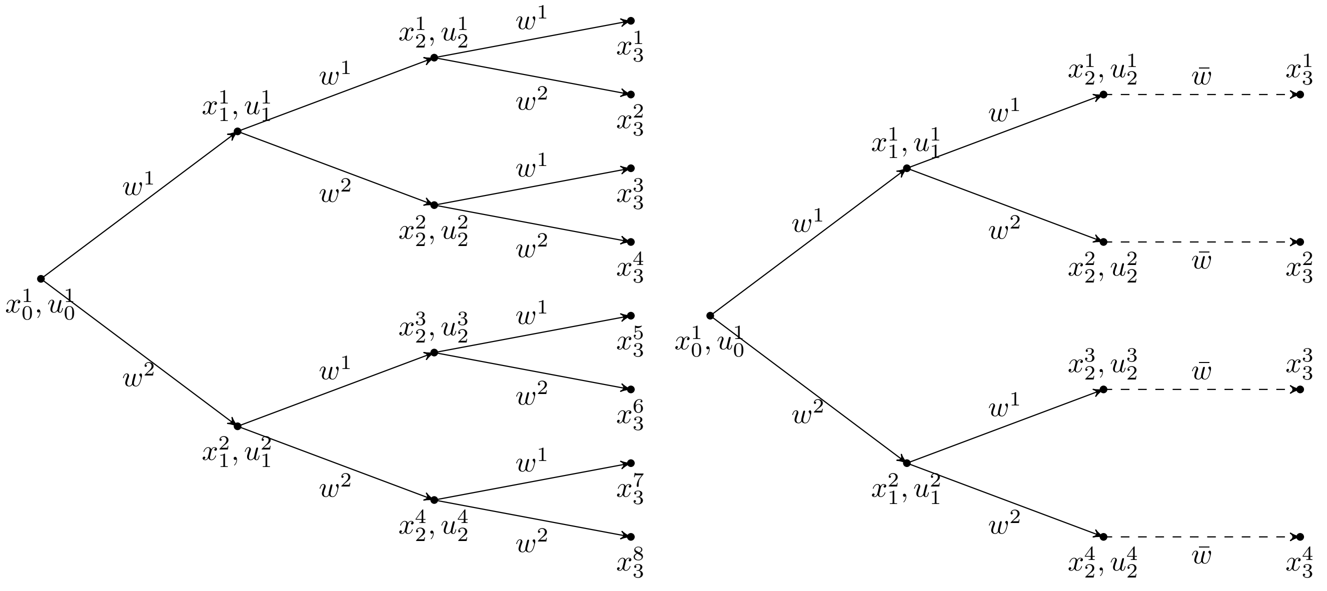


Figure 1: Right: Visualization of a scenario tree split at every node. Right: Visualization of scenario tree, split only up to a robust horizon N_{rob} (here: $N_{\text{rob}} = 2$)

Multistage robust OCP (scenario-tree robust OCP)

We now consider a multistage (or scenario-tree) robust OCP. In order to construct the scenario tree, we must first discretize the disturbance set $\mathcal{E}(0, I)$. Here, we consider the discrete set $\mathcal{W} = \{0, (1,0), (-1,0), (0,1), (0,-1)\} = \{w^1, \dots, w^m\}$, yielding $m = |\mathcal{W}| = 5$ possible disturbance values per discrete time point. For the node numbering convention of the resulting scenario tree, we refer to Fig. 1 The tree OCP can be written as

$$\min_{x,\,u} \quad \sum_{k=0}^{N-1} \left(\frac{1}{m^k} \sum_{i=1}^{m^k} l(x_k^i, u_k^i) \right) + \frac{1}{m^N} \sum_{i=1}^{m^N} l_N(x_N^i) \tag{10a}$$

$$\text{s.t.} \quad x_0^0 = \bar{\bar{x}}_0, \tag{10b}$$

$$x_{k+1}^i = f(x_k^{\lceil i/m^k \rceil}, u_k^{\lceil i/m^k \rceil}, w^{i]_1^m}), \qquad k = 0, \dots, N-1,\ i = 1, \dots, m^{k+1}, \tag{10c}$$

$$0 \le h_k(x_k^i, u_k^i), \qquad k = 0, \dots, N-1,\ i = 1, \dots, m^k, \tag{10d}$$

$$0 \le h_N(x_k^i, u_k^i), \qquad i = 1, \dots, m^N, \tag{10e}$$

where $\lceil \cdot \rceil$ denotes the ceiling function and $i]_1^m$ wraps the integer i to the set $\{1, \dots, m\}$. Thus, for each $k = 0, \dots, N-1$, the dynamics constraint (10c) cycles through all scenarios of the current stage, (x_k^i, u_k^i), $i = 1, \dots, m^k$, and simulates it forward once for every possible disturbance value, $w^i \in \mathcal{W}$.

Tasks

1. Take a moment to consider the complexity of the tree OCP. Given our specific values of N and m, how many terminal nodes does the resulting tree have?

2. The file `solver_tree_ocp.py` implements a tree OCP. In order to alleviate the exponential scenario growth, it allows to set a value for the robust horizon N_{rob}, up to which the disturbances are considered. For the remainder of the OCP horizon, each scenario is propagated only nominally, cf. the righthand side of Fig. 1.
   Complete the template, and run the tree solver via `main.py`. Compare the resulting trajectories to the solutions from the previous OCPs.

3. Try increasing the value of N_{rob}. How far can you go while keeping within reasonable computation times?

Bonus tasks

1. Already start working on the next exercise sheet.
2. Apply the above robust OCP solvers to your favorite dynamical system.
3. Create a solver class that optimizes a time-variant feedback law parametrization.