Dufour MPC formulation
ACADOS

Jonas Ohnemus

December 8, 2023

Contents

Chapter 1

Introduction

The goal of this season in the control module is to switch to ROS2, clean up the codebase to not have several controllers for the same task, and deal with the issue of not necessarily having a track known before the driverless trackdrive discipline on the FSG competition.

1.1 Problem Setting

In AMZ, the DV-controls module is responsible for the development, testing, and deployment of high-level control approaches. We build the algorithms that decide on which steering angle and longitudinal acceleration is needed given estimates of cone positions that define the track (perception) and the vehicle state within this track (estimation). Our signals are then subsequently processed by lower-level controllers that, for example, regulate motor torques directly. To this extent, the controls module takes the current state of the car (pose, velocities, actuator states, ...) and state of the environment (cone positions which define track boundaries, ...), possibly predicts ahead what the car could do, and finally decides which actions the driverless vehicle should take to get the car over the finish line the fastest while respecting several constraints.

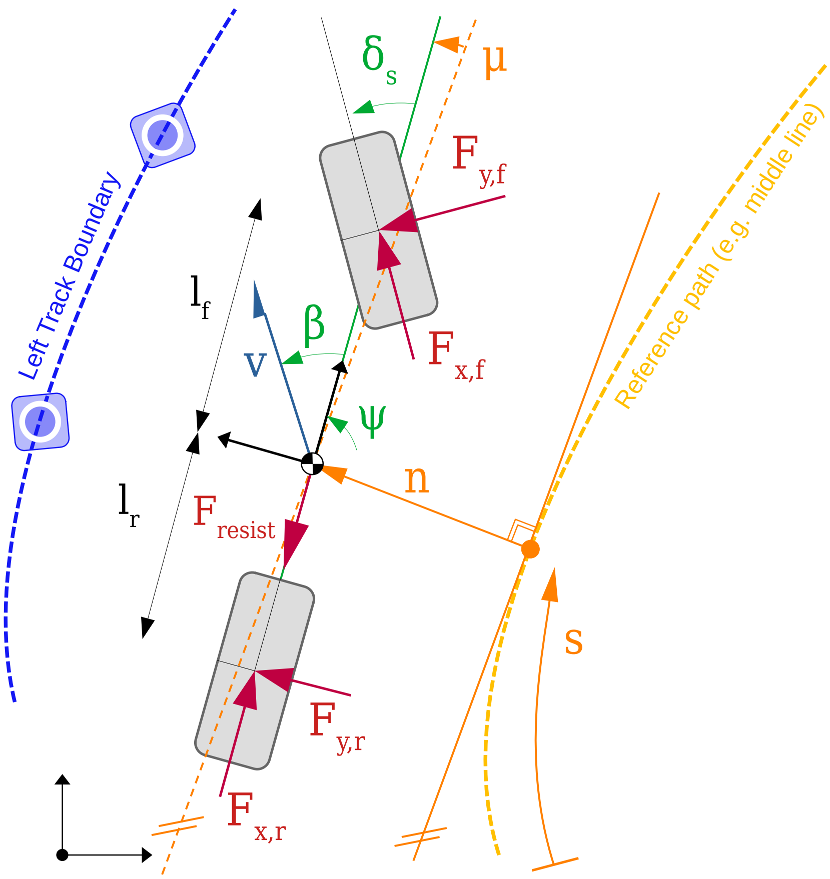

Figure 1.1: Dynamic Bicycle Model in curvilinear coordinates

1.2 Control Approach Decision

Due to the importance of constraint satisfaction, and because this is one of the outstanding features of Model Predictive Control, we have decided to implement MPC for longitudinal and lateral vehicle control.

Chapter 2

MPC Formulation: implementation using ACADOS solver

Although AMZ has maintained a good relationship to Embotech, which is a company that provides real-time nonlinear MPC solvers, we wanted to see if the Open Source project, ACADOS, can provide equally good solvers for our application. To this end, we will now formulate the MPC optimal control problem for the autonomous driving task in the ACADOS framework given in Appendix A.

2.1 Model

2.1.1 Dynamic Model

The state vector of the MPC model is given by:

$$\mathbf{x}(t) = \begin{bmatrix} s \\ n \\ \mu \\ {}_V v_x \\ {}_V v_y \\ \dot{\psi} \\ F_{x,m} \\ \delta_s \end{bmatrix} \tag{2.1}$$

And the ODE then subsequently as:

$$f_{expl,\,dyn}(\mathbf{x},\mathbf{u}) = \begin{bmatrix} \dot{s} \\ \dot{n} \\ \dot{\mu} \\ {}_V \dot{v}_x \\ {}_V \dot{v}_y \\ \ddot{\psi} \\ \dot{F}_{x,m} \\ \dot{\delta}_s \end{bmatrix} = \begin{bmatrix} \frac{{}_V v_x \cos(\mu) - {}_V v_y \sin(\mu)}{1 - n\kappa(s)} \\ {}_V v_x \sin(\mu) + {}_V v_y \cos(\mu) \\ \dot{\psi} - \kappa(s)\, \frac{{}_V v_x \cos(\mu) - {}_V v_y \sin(\mu)}{1 - n\kappa(s)} \\ \frac{1}{m}\left(\frac{1}{2} F_{x,m}\left[1 + \cos(\delta_s)\right] - F_{y,f}\sin(\delta_s) - F_{resist}\right) + \dot{\psi} \cdot {}_V v_y \\ \frac{1}{m}\left(F_{y,f}\cos(\delta_s) + F_{y,r} + \frac{1}{2} F_{x,m}\sin(\delta_s)\right) - \dot{\psi} \cdot {}_V v_x \\ \frac{1}{I_z}\left(\frac{1}{2} F_{x,m}\sin(\delta_s) l_f + F_{y,f}\cos(\delta_s) l_f - F_{y,r} l_r\right) \\ \dot{F}_{x,m,u} \\ \dot{\delta}_{s,u} \end{bmatrix} \tag{2.2}$$

subject to:

Resistive Force

$$F_{x,fric} = C_r \tag{2.3}$$

$$F_{x,drag} = C_d \cdot {}_V v_x^2 \tag{2.4}$$

$$F_{resist} = F_{x,fric} + F_{x,drag} \tag{2.5}$$

Slip Angles

$$\alpha_f = \arctan 2\left(\frac{{}_V v_y + \dot{\psi} l_f}{{}_V v_x}\right) - \delta_s \qquad \alpha_r = \arctan 2\left(\frac{{}_V v_y - \dot{\psi} l_r}{{}_V v_x}\right) \tag{2.6}$$

Lateral Pacejka Magic Formula

$$F_{y,f} = F_{z,f}\, D_{tire} \sin\left(C_{tire} \arctan(B_{tire}\alpha_f)\right) \tag{2.7}$$

$$F_{y,r} = F_{z,r}\, D_{tire} \sin\left(C_{tire} \arctan(B_{tire}\alpha_r)\right) \tag{2.8}$$

Longitudinal and lateral accelerations (in the vehicle frame)

$$ {}_V a_x = {}_V \dot{v}_x - \dot{\psi} \cdot {}_V v_y \tag{2.9} $$

$$ {}_V a_y = {}_V \dot{v}_y + \dot{\psi} \cdot {}_V v_x \tag{2.10} $$

Vertical (Normal) Forces

$$ F_{z,f} = mg \frac{l_r}{l_r + l_f} \tag{2.11} $$

$$ F_{z,r} = mg \frac{l_f}{l_r + l_f} \tag{2.12} $$

2.1.2 Kinematic Model

For the kinematic model, we only adjust the right hand sides of the explicit dynamic ODE described before for ${}_V v_x$, ${}_V v_y$, and $\dot{\psi}$.

$$ f_{expl,\,kin}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{s} \\ \dot{n} \\ \dot{\mu} \\ {}_V \dot{v}_x \\ {}_V \dot{v}_y \\ \ddot{\psi} \\ \dot{F}_{x,m} \\ \dot{\delta}_s \end{bmatrix} = \begin{bmatrix} \frac{{}_V v_x \cos(\mu) - {}_V v_y \sin(\mu)}{1 - n\kappa(s)} \\ {}_V v_x \sin(\mu) + {}_V v_y \cos(\mu) \\ \dot{\psi} - \kappa(s) \, \frac{{}_V v_x \cos(\mu) - {}_V v_y \sin(\mu)}{1 - n\kappa(s)} \\ \frac{1}{m}\left(F_{x,m} - F_{resist}\right) \\ \left(\dot{\delta}_{s,u} \cdot {}_V v_x + \delta_s \cdot \frac{1}{m} F_{x,m}\right) \cdot \frac{l_r}{l_r + l_f} \\ \left(\dot{\delta}_{s,u} \cdot {}_V v_x + \delta_s \cdot \frac{1}{m} F_{x,m}\right) \cdot \frac{1}{l_r + l_f} \\ \dot{F}_{x,m,u} \\ \dot{\delta}_{s,u} \end{bmatrix} \tag{2.13} $$

2.1.3 Model Combination

Given a kinematic model and a dynamic model, we blend between them by introducing a blending factor $\gamma_{blend} \in [0,\ 1]$ and computing the convex combination of both right hand sides.

$$ \dot{\mathbf{x}}(t) = f_{expl}(\mathbf{x}, \mathbf{u}) = \gamma_{blend} \cdot f_{expl,\,dyn}(\mathbf{x}, \mathbf{u}) + (1 - \gamma_{blend}) \cdot f_{expl,\,kin}(\mathbf{x}, \mathbf{u}) \tag{2.14} $$

Switching between the models is then done based on e.g. lateral acceleration or longitudinal velocity.

2.1.4 Algebraic States

Since acados lets us introduce algebraic variables $\mathbf{z}(t)$, we make use of them to store stage-dependent information that would otherwise not occur in the state. One example is the reference path curvature. Given a (at compile time) fixed vector $\mathbf{s}_{ref} \in \mathbb{R}^{n_\kappa}$, we define a differentiable lookup table from the current state s to $\kappa_{LUT}(s)$, where $\kappa_{ref} \in \mathbb{R}^{n_\kappa}$ is the corresponding vector for the curvature values ahead and is defined in the ROS-node as a parameter to the MPC. Finally, the slip angles are also defined as algebraic states for easy access.
The algebraic state vector follows as

$$ \mathbf{z}(t) = f_{expl,\mathbf{z}}(\mathbf{x}, \mathbf{u}) \begin{bmatrix} \kappa_{LUT} \\ \alpha_f \\ \alpha_r \end{bmatrix} \tag{2.15} $$

2.1.5 The full dynamic model

Finally, we concatenate the differential and algebraic state vectors and make use of the implicit dynamics formulation of acados. Here we also make the dependence on the parameters $\mathbf{p}$ explicit.

$$0 = f_{impl}(\mathbf{x}(t), \dot{\mathbf{x}}(t), \mathbf{u}(t), \mathbf{z}(t), \mathbf{p}) = \begin{bmatrix} \dot{\mathbf{x}}(t) - f_{expl}(\mathbf{x}, \mathbf{u}, \mathbf{p}) \\ \mathbf{z}(t) - f_{expl,\mathbf{z}}(\mathbf{x}, \mathbf{u}) \end{bmatrix} \tag{2.16}$$

2.2 Inequality Constraints

2.2.1 Nonlinear Stage Constraints

We define in total 8 nonlinear inequality constraints, of which 7 are slacked.
The first two are tire ellipse constraints for the front and rear wheel respectively:

$$\left(\frac{F_{y,f}}{F_{z,f} D_{tire} \epsilon_{y,f}}\right)^2 + \left(\frac{F_{x,m}}{2} \cdot \frac{\epsilon_{x,f}}{F_{z,f} D_{tire} \epsilon_{y,f}}\right)^2 - 1 \leq s_{u,h,TE,f} \tag{2.17}$$

$$\left(\frac{F_{y,r}}{F_{z,r} D_{tire} \epsilon_{y,r}}\right)^2 + \left(\frac{F_{x,m}}{2} \cdot \frac{\epsilon_{x,r}}{F_{z,r} D_{tire} \epsilon_{y,r}}\right)^2 - 1 \leq s_{u,h,TE,r} \tag{2.18}$$

where $D_{tire} \epsilon_{y,f} = \epsilon_{max,f}$ and $D_{tire} \epsilon_{y,r} = \epsilon_{max,r}$.
Following the tire constraints, we have four nonlinear inequalities that, in a slacked fashion, make sure the car stays on the track. Based on the heading difference angle μ, we first define three bound variables corresponding to the front, rear, and width dimension of the car.

$$b_f = L_F \cdot \sin(\mu) \tag{2.19}$$

$$b_r = L_R \cdot \sin(\mu) \tag{2.20}$$

$$b_w = \frac{W}{2} \cdot \cos(\mu) \tag{2.21}$$

From these definitions, we can formulate the following three inequality constraints, where $n_{\min} < 0$ and $n_{\max} > 0$ define the track bounds (possibly based on s?).

$$n + b_f + b_w - n_{\max} \leq s_{u,h,n,fl} \tag{2.22}$$

$$n - b_f + b_w - n_{\max} \leq s_{u,h,n,rl} \tag{2.23}$$

$$-n - b_f + b_w + n_{\min} \leq s_{u,h,n,fr} \tag{2.24}$$

$$-n + b_f + b_w + n_{\min} \leq s_{u,h,n,rr} \tag{2.25}$$

The seventh nonlinear inequality constraint is **not slacked**. It relates the curvature with the lateral deviation from track.

$$\kappa \cdot n - 1 \leq 0 \tag{2.26}$$

To give some sort of bound on velocity along the planning stages, we introduce a (weakly) slacked upper bound on the longitudinal velocity ${}_V v_x$.

$${}_V v_x - {}_V v_{x,\max} \leq s_{u,h,vx,rr} \tag{2.27}$$

2.2.2 Nonlinear Terminal Constraints

Except for one upper bound, the terminal constraints are the same as the nonlinear stage constraints. The terminal velocity is reduced for this stage as we want to prevent too optimistic planning which could reduce our recursive feasibility properties.

2.2.3 Linear Stage and Terminal Constraints

For the linear stage and terminal constraints, we only loosely restrict the state variables.

2.3 Cost function

$$J_{MPC}(\mathbf{x}_t, \mathbf{u}_t) = -\dot{s}_t + (\mathbf{x} - \mathbf{x}_{ref})^T Q (\mathbf{x} - \mathbf{x}_{ref}) + \mathbf{u}^T R \mathbf{u} + S(\mathbf{x}) \tag{2.28}$$

Appendix A

acados OCP (Optimal Control Problem) Formulation

A.1 Problem Formulation

acados can handle the following optimization problem

$$
\begin{aligned}
&\min_{x(\cdot),u(\cdot),z(\cdot),s(\cdot),s^{\mathrm{e}}} && \texttt{/* Cost function, see section A.3 */} \\
& && \int_0^T l(x(\tau),u(\tau),z(\tau),p) + \frac{1}{2}\begin{bmatrix} s_{\mathrm{l}}(\tau) \\ s_{\mathrm{u}}(\tau) \\ 1 \end{bmatrix}^\top \begin{bmatrix} Z_{\mathrm{l}} & 0 & z_{\mathrm{l}} \\ 0 & Z_{\mathrm{u}} & z_{\mathrm{u}} \\ z_{\mathrm{l}}^\top & z_{\mathrm{u}}^\top & 0 \end{bmatrix} \begin{bmatrix} s_{\mathrm{l}}(\tau) \\ s_{\mathrm{u}}(\tau) \\ 1 \end{bmatrix} \mathrm{d}\tau + \\
& && m(x(T),z(T),p) + \frac{1}{2}\begin{bmatrix} s^{\mathrm{e}}_{\mathrm{l}} \\ s^{\mathrm{e}}_{\mathrm{u}} \\ 1 \end{bmatrix}^\top \begin{bmatrix} Z^{\mathrm{e}}_{\mathrm{l}} & 0 & z^{\mathrm{e}}_{\mathrm{l}} \\ 0 & Z^{\mathrm{e}}_{\mathrm{u}} & z^{\mathrm{e}}_{\mathrm{u}} \\ z^{\mathrm{e}\top}_{\mathrm{l}} & z^{\mathrm{e}\top}_{\mathrm{u}} & 0 \end{bmatrix} \begin{bmatrix} s^{\mathrm{e}}_{\mathrm{l}} \\ s^{\mathrm{e}}_{\mathrm{u}} \\ 1 \end{bmatrix} && \text{(A.1)} \\
& && \texttt{/* Initial values, see section A.4.1 */} \\
&\text{s.t.} && \underline{x}_0 \le J_{\mathrm{bx},0}\, x(0) \le \bar{x}_0, && \text{(A.2)} \\
& && \texttt{/* Dynamics, see section A.2 */} \\
& && f_{\mathrm{impl}}(x(t),\dot{x}(t),u(t),z(t),p) = 0, && t\in[0,T), \quad \text{(A.3)} \\
& && \texttt{/* Path constraints with lower bounds, see section A.4.2 */} \\
& && \underline{h} \le h(x(t),u(t),p) + J_{\mathrm{sh}}\, s_{\mathrm{l,h}}(t), && t\in[0,T), \quad \text{(A.4)} \\
& && \underline{x} \le J_{\mathrm{bx}}\, x(t) + J_{\mathrm{sbx}}\, s_{\mathrm{l,bx}}(t), && t\in(0,T), \quad \text{(A.5)} \\
& && \underline{u} \le J_{\mathrm{bu}}\, u(t) + J_{\mathrm{sbu}}\, s_{\mathrm{l,bu}}(t), && t\in[0,T), \quad \text{(A.6)} \\
& && \underline{g} \le C\, x(t) + D\, u(t) + J_{\mathrm{sg}}\, s_{\mathrm{l,g}}(t), && t\in[0,T), \quad \text{(A.7)} \\
& && s_{\mathrm{l,h}}(t), s_{\mathrm{l,bx}}(t), s_{\mathrm{l,bu}}(t), s_{\mathrm{l,g}}(t) \ge 0, && t\in[0,T), \quad \text{(A.8)} \\
& && \texttt{/* Path constraints with upper bounds, see section A.4.2 */} \\
& && h(x(t),u(t),p) - J_{\mathrm{sh}}\, s_{\mathrm{u,h}}(t) \le \bar{h}, && t\in[0,T), \quad \text{(A.9)} \\
& && J_{\mathrm{bx}} x(t) - J_{\mathrm{sbx}}\, s_{\mathrm{u,bx}}(t) \le \bar{x}, && t\in(0,T), \text{(A.10)} \\
& && J_{\mathrm{bu}} u(t) - J_{\mathrm{sbu}}\, s_{\mathrm{u,bu}}(t) \le \bar{u}, && t\in[0,T), \text{(A.11)} \\
& && C x(t) + D u(t) - J_{\mathrm{sg}}\, s_{\mathrm{u,g}} \le \bar{g}, && t\in[0,T), \text{(A.12)} \\
& && s_{\mathrm{u,h}}(t), s_{\mathrm{u,bx}}(t), s_{\mathrm{u,bu}}(t), s_{\mathrm{u,g}}(t) \ge 0, && t\in[0,T), \text{(A.13)} \\
& && \texttt{/* Terminal constraints with lower bounds, see section A.4.3 */} \\
& && \underline{h}^{\mathrm{e}} \le h^{\mathrm{e}}(x(T),p) + J^{\mathrm{e}}_{\mathrm{sh}}\, s^{\mathrm{e}}_{\mathrm{l,h}}, && \text{(A.14)} \\
& && \underline{x}^{\mathrm{e}} \le J^{\mathrm{e}}_{\mathrm{bx}}\, x(T) + J^{\mathrm{e}}_{\mathrm{sbx}}\, s^{\mathrm{e}}_{\mathrm{l,bx}}, && \text{(A.15)} \\
& && \underline{g}^{\mathrm{e}} \le C^{\mathrm{e}}\, x(T) + J^{\mathrm{e}}_{\mathrm{sg}}\, s^{\mathrm{e}}_{\mathrm{l,g}} \le \bar{g}^{\mathrm{e}}, && \text{(A.16)} \\
& && s^{\mathrm{e}}_{\mathrm{l,h}}, s^{\mathrm{e}}_{\mathrm{l,bx}}, s^{\mathrm{e}}_{\mathrm{l,bu}}, s^{\mathrm{e}}_{\mathrm{l,g}} \ge 0, && \text{(A.17)} \\
& && \texttt{/* Terminal constraints with upper bound, see section A.4.3 */} \\
& && h^{\mathrm{e}}(x(T),p) - J^{\mathrm{e}}_{\mathrm{sh}}\, s^{\mathrm{e}}_{\mathrm{u,h}} \le \bar{h}^{\mathrm{e}}, && \text{(A.18)} \\
& && J^{\mathrm{e}}_{\mathrm{bx}}\, x(T) - J^{\mathrm{e}}_{\mathrm{sbx}}\, s^{\mathrm{e}}_{\mathrm{u,bx}} \le \bar{x}^{e}, && \text{(A.19)} \\
& && C^{\mathrm{e}}\, x(T) - J^{\mathrm{e}}_{\mathrm{sg}}\, s^{\mathrm{e}}_{\mathrm{u,g}} \le \bar{g}^{\mathrm{e}} && \text{(A.20)} \\
& && s^{\mathrm{e}}_{\mathrm{u,h}}, s^{\mathrm{e}}_{\mathrm{u,bx}}, s^{\mathrm{e}}_{\mathrm{u,bu}}, s^{\mathrm{e}}_{\mathrm{u,g}} \ge 0, && \text{(A.21)}
\end{aligned}
$$

with

- state vector $x : \mathbb{R} \to \mathbb{R}^{n_x}$

- control vector $u : \mathbb{R} \to \mathbb{R}^{n_\mathrm{u}}$
- algebraic state vector $z : \mathbb{R} \to \mathbb{R}^{n_\mathrm{z}}$
- model parameters $p \in \mathbb{R}^{n_\mathrm{p}}$
- slacks for path constraints $s_\mathrm{l}(t) = (s_\mathrm{l,bu}, s_\mathrm{l,bx}, s_\mathrm{l,g}, s_\mathrm{l,h}) \in \mathbb{R}^{n_\mathrm{s}}$ and $s_\mathrm{u}(t) = (s_\mathrm{u,bu}, s_\mathrm{u,bx}, s_\mathrm{u,g}, s_\mathrm{u,h}) \in \mathbb{R}^{n_\mathrm{s}}$
- slacks for terminal constraints $s_\mathrm{l}^\mathrm{e}(t) = (s_\mathrm{l,bx}^\mathrm{e}, s_\mathrm{l,g}^\mathrm{e}, s_\mathrm{l,h}^\mathrm{e}) \in \mathbb{R}^{n_\mathrm{s}^\mathrm{e}}$ and $s_\mathrm{u}^\mathrm{e}(t) = (s_\mathrm{u,bx}^\mathrm{e}, s_\mathrm{u,g}^\mathrm{e}, s_\mathrm{u,h}^\mathrm{e}) \in \mathbb{R}^{n_\mathrm{s}^\mathrm{e}}$

Some of the following restrictions may apply to matrices in the formulation:

DIAG	diagonal
SPUM	horizontal slice of a permuted unit matrix
SPUME	like **SPUM**, but with empty rows intertwined

Document Purpose This document is only associated to the Matlab interface of acados. Here, the focus is to give a mathematical overview of the problem formulation and possible options to model it within acados. The problem formulation and the possibilities of acados are similar in the Python interface, however, some of the string identifiers are different. The documentation is not exhaustive and does not contain a full description for the Matlab interface.

You can find examples int the directory `<acados>/examples/acados_matlab_octave`. The source code of the acados Matlab interface is found in: `<acados>/interfaces/acados_matlab_octave` and should serve as a more extensive, complete and up-to-date documentation about the possibilities.

A.2 Dynamics

The system dynamics term is used to connect state trajectories from adjacent shooting nodes by means of equality constraints. The system dynamics equation (A.3) is replaced with a discrete-time dynamic system. The dynamics can be formulated in different ways in acados: As implicit equations in continuous time (A.22), or as explicit equations in continuous time (A.23) or directly as discrete-time dynamics (A.24). This section and table A.1 summarizes the options.

A.2.1 Implicit Dynamics

The most general way to provide a continuous time ODE in acados is to define the function $f_\mathrm{impl} : \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_\mathrm{u}} \times \mathbb{R}^{n_\mathrm{z}} \times \mathbb{R}^{n_\mathrm{p}} \to \mathbb{R}^{n_\mathrm{x}+n_\mathrm{z}}$ which is fully implicit DAE formulation describing the system as:

$$f_\mathrm{impl}(x, \dot{x}, u, z, p) = 0. \tag{A.22}$$

acados can discretize f_impl with a classical implicit Runge-Kutta (`irk`) or a structure exploiting implicit Runge-Kutta method (`irk_gnsf`). Both discretization methods are set using the `'sim_method'` identifier in a `acados_ocp_opts` class instance.

A.2.2 Explicit Dynamics

Alternatively, acados offers an explicit Runge-Kutta integrator (`erk`), which can be used with explicit ODE models, i.e., models of the form

$$f_\mathrm{expl}(x, u, p) = \dot{x}. \tag{A.23}$$

A.2.3 Discrete Dynamics

Another option is to provide a discrete function that maps state x_i, control u_i and parameters p_i from shooting node i to the state x_{i+1} of the next shooting node $i+1$, i.e., a function

$$x_{i+1} = f_\mathrm{disc}(x_i, u_i, p_i). \tag{A.24}$$

Table A.1: Dynamics definitions

Term	String identifier	Data type	Required
f_{impl} respectively f_{expl}	`dyn_expr_f`	`CasADi` expression	yes
f_{disc}	`dyn_exp_phi`	`CasADi` expression	yes
-	`dyn_type`	string (`'explicit'`, `'implicit'` or `'discrete'`)	yes

A.3 Cost

There are different acados modules to model the cost functions in equation (A.1).

- $l : \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_\mathrm{u}} \times \mathbb{R}^{n_\mathrm{z}} \to \mathbb{R}$ is the Lagrange objective term.
- $m : \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_\mathrm{z}} \to \mathbb{R}$ is the Mayer objective term.

to define which one is used set `cost_type` for l, `cost_type_e` for m.

Setting the slack penalties in equation (A.1) is done in the same way for all cost modules, see table A.2 for an overview. Moreover, you can specify `cost_Z`, to set Z_l, Z_u to the same values, i.e., use a symmetric L2 slack penalty. Similarly,

Table A.2: Cost module slack variable options

Term	String id	Data type	Required
Z_l	`cost_Zl`	double, **DIAG**	no
Z_u	`cost_Zu`	double, **DIAG**	no
z_l	`cost_zl`	double	no
z_u	`cost_zu`	double	no
Z_l^e	`cost_Zl_e`	double, **DIAG**	no
Z_u^e	`cost_Zu_e`	double, **DIAG**	no
z_l^e	`cost_zl_e`	double	no
z_u^e	`cost_zu_e`	double	no

`cost_z`, `cost_Z_e`, `cost_z_e` can be used to set symmetric slack L1 penalties, respectively penalties for the terminal slack variables.

Note, that the dimensions of the slack variables $s_\mathrm{l}(t)$, $s_\mathrm{l}^\mathrm{e}(t)$, $s_\mathrm{u}(t)$ and $s_\mathrm{u}^\mathrm{e}(t)$ are determined by acados from the associated matrices (Z_l, Z_u, J_sh, J_sg, J_sbu, J_sbx etc.).

A.3.1 Cost module: `auto`

Set `cost_type` to `auto` (default). In this case acados detects if the cost function specified is a linear least squares term and transcribes it in the corresponding form. Otherwise, it is formulated using the external cost module. Note: slack penalties are optional and we plan to detected them from the expressions in future versions. Table A.3 shows the available options.

Table A.3: Cost module `auto` options

Term	String identifier	Data type	Required
l	`cost_expr_ext_cost`	`CasADi` expression	yes

A.3.2 Cost module: `external`

Set `cost_type` to `ext_cost`. See table A.4 for the available options.

Table A.4: Cost module `external` options

Term	String identifier	Data type	Required
l	`cost_expr_ext_cost`	`CasADi` expression	yes
m	`cost_expr_ext_cost_e`	`CasADi` expression	yes

A.3.3 Cost module: `linear least squares`

In order to activate the `linear least squares` cost module, set `cost_type` to `linear_ls`.
The Lagrange cost term has the form

$$l(x,u,z) = \frac{1}{2}\left\|\underbrace{V_x\,x + V_u\,u + V_z\,z}_{y} - y_{\text{ref}}\right\|_W^2 \tag{A.25}$$

where matrices $V_x \in \mathbb{R}^{n_y \times n_x}$, $V_u \in \mathbb{R}^{n_y \times n_u}$ are $V_z \in \mathbb{R}^{n_y \times n_z}$ map x, u and z onto y, respectively and $W \in \mathbb{R}^{n_y \times n_y}$ is the weighing matrix. The vector $y_{\text{ref}} \in \mathbb{R}^{n_y}$ is the reference.

Similarly, the Mayer cost term has the form

$$m(x,u,z) = \frac{1}{2}\left\|\underbrace{V_x^{\text{e}} x}_{y^{\text{e}}} - y_{\text{ref}}^{\text{e}}\right\|_{W^{\text{e}}}^2 \tag{A.26}$$

where matrix $V_x^{\text{e}} \in \mathbb{R}^{n_{y^{\text{e}}} \times n_x}$ maps x onto y^{e} and $W^{\text{e}} \in \mathbb{R}^{n_{y^{\text{e}}} \times n_{y^{\text{e}}}}$ is the weighing matrix. The vector $y_{\text{ref}}^{\text{e}} \in \mathbb{R}^{n_{y^{\text{e}}}}$ is the reference.

See table A.5 for the available options of this cost module.

Table A.5: Cost module `linear_ls` options

Term	String identifier	Data type	Required
V_x	`cost_Vx`	double	yes
V_u	`cost_Vu`	double	yes
V_z	`cost_Vz`	double	yes
W	`cost_W`	double	yes
y_{ref}	`cost_y_ref`	double	yes
V_x^{e}	`cost_Vx_e`	double	yes
W^{e}	`cost_W_e`	double	yes
$y_{\text{ref}}^{\text{e}}$	`cost_y_ref_e`	double	yes

A.3.4 Cost module: `nonlinear least squares`

In order to activate the `nonlinear least squares` cost module, set `cost_type` to `nonlinear_ls`.

The `nonlinear least squares` cost function has the same basic form as eqns. (A.25 - A.26) of the `linear least squares` cost module. The only difference is that y and y^{e} are defined by means of `CasADi` expressions, instead of via matrices V_x, V_u, V_z and V_x^{e}. See table A.6 for the available options of this cost module.

Table A.6: Cost module `nonlinear_ls` options

Term	String identifier	Data type	Required
y	`cost_expr_y`	`CasADi` expression	yes
W	`cost_W`	double	yes
y_{ref}	`cost_y_ref`	double	yes
y^{e}	`cost_expr_y_e`	`CasADi` expression	yes
W^{e}	`cost_W_e`	double	yes
y^{e}_{ref}	`cost_y_ref_e`	double	yes

A.4 Constraints

This section is about how to define the constraints equations (A.2) and (A.4 - A.21).

The Matlab interface supports the constraint module bgh, which is able to handle simple **b**ounds (on x and u), **g**eneral linear constraints and general nonlinear constraints. Meanwhile, the Python interface also supports the acados constraint module bgp, which can handle convex-over-nonlinear constraints in a dedicated fashion.

A.4.1 Initial State

Note: An initial state is not required. For example for moving horizon estimation (MHE) problems it should not be set.

Two possibilities exist to define the initial states equation (A.2): a simple syntax and an extended syntax.

Simple syntax defines the full initial state $x(0) = \bar{x}_0$. The options are found in table A.7.

Table A.7: Simple syntax for setting the initial state

Term	String identifier	Data type	Required
$\bar{x}_0$	`constr_x0`	double	no

Extended syntax allows to define upper and lower bounds on a subset of states. The options for the extended syntax are found in table A.8.

Table A.8: Extended syntax for setting the initial state

Term	String identifier	Data type	Required
$\underline{x}_0$	`constr_lbx_0`	double	no
$\bar{x}_0$	`constr_ubx_0`	double	no
$J_{\text{bx},0}$	`constr_Jbx_0`	double	no

A.4.2 Path Constraints

Table A.9 shows the options for defining the path constraints equations (A.4 - A.13). Here, matrices

- J_{sh}, maps lower slack vectors $s_{\text{l,h}}(t)$ and upper slack vectors $s_{\text{u,h}}(t)$ onto the non-linear constraint expressions $h(x,u,p)$.

- J_{bx}, J_{bu} map $x(t)$ and $u(t)$ onto its bounds vectors $\underline{x}$, $\bar{x}$ and $\underline{u}$, $\bar{u}$, respectively.
- J_{sx}, J_{su} map lower slack vectors $s_{\mathrm{l,bx}}(t)$, $s_{\mathrm{l,bu}}(t)$ and upper slack vectors $s_{\mathrm{u,bx}}(t)$, $s_{\mathrm{u,bu}}(t)$ onto $x(t)$ and $u(t)$, respectively.
- J_{sg} map lower slack vectors $s_{\mathrm{l,g}}(t)$ and upper slack vectors $s_{\mathrm{u,g}}(t)$ onto lower and upper equality bounds $\underline{g}$, $\bar{g}$, respectively.
- C, D map $x(t)$ and $u(t)$ onto lower and upper inequality bounds $\underline{g}$, $\bar{g}$ (polytopic constraints)

Table A.9: Path constraints options

Term	String identifier	Data type	Required
J_{bx}	`constr_Jbx`	double, **SPUM**	no
$\underline{x}$	`constr_lbx`	double	no
$\bar{x}$	`constr_ubx`	double	no
J_{bu}	`constr_Jbu`	double, **SPUM**	no
$\underline{u}$	`constr_lbu`	double	no
$\bar{u}$	`constr_ubu`	double	no
C	`constr_C`	double	no
D	`constr_D`	double	no
$\underline{g}$	`constr_lg`	double	no
$\bar{g}$	`constr_ug`	double	no
h	`constr_expr_h`	`CasADi` expression	no
$\underline{h}$	`constr_lh`	double	no
$\bar{h}$	`constr_uh`	double	no
J_{sbx}	`constr_Jsbx`	double, **SPUME**	no
J_{sbu}	`constr_Jsbu`	double, **SPUME**	no
J_{sg}	`constr_Jsg`	double, **SPUME**	no
J_{sbx}	`constr_Jsh`	double, **SPUME**	no

A.4.3 Terminal Constraints

Table A.10 shows the options for defining the terminal constraints equations (A.14 - A.21). Here, matrices

- $J^{\mathrm{e}}_{\mathrm{sh}}$, maps lower slack vectors $s^{\mathrm{e}}_{\mathrm{l,h}}(t)$ and upper slack vectors $s^{\mathrm{e}}_{\mathrm{u,h}}(t)$ onto non-linear terminal constraint expressions $h^{\mathrm{e}}(x(T), p)$.
- $J^{\mathrm{e}}_{\mathrm{bx}}$ maps $x(T)$ onto its bounds vectors $\underline{x}^{\mathrm{e}}$ and $\bar{x}^{\mathrm{e}}$.
- $J^{\mathrm{e}}_{\mathrm{sbx}}$ maps lower slack vectors $s^{\mathrm{e}}_{\mathrm{l,bx}}$ and upper slack vectors $s^{\mathrm{e}}_{\mathrm{u,bx}}$ onto $x(T)$.
- $J^{\mathrm{e}}_{\mathrm{sg}}$ map lower slack vectors $s^{\mathrm{e}}_{\mathrm{l,g}}(t)$ and upper slack vectors $s^{\mathrm{e}}_{\mathrm{u,g}}(t)$ onto lower and upper equality bounds $\underline{g}^{\mathrm{e}}$, $\bar{g}^{\mathrm{e}}$, respectively.
- C^{e} maps $x(T)$ onto lower and upper inequality bounds $\underline{g}^{\mathrm{e}}$, $\bar{g}^{\mathrm{e}}$ (polytopic constraints)

Table A.10: Terminal constraints options

Term	String identifier	Data type	Required
J_{bx}^{e}	`constr_Jbx_e`	double, **SPUM**	no
$\underline{x}^{e}$	`constr_lbx_e`	double	no
$\bar{x}^{e}$	`constr_ubx_e`	double	no
C^{e}	`constr_C_e`	double	no
$\underline{g}^{e}$	`constr_lg`	double	no
$\bar{g}^{e}$	`constr_ug`	double	no
h^{e}	`constr_expr_h_e`	`CasADi` expression	no
$\underline{h}^{e}$	`constr_lh_e`	double	no
$\bar{h}^{e}$	`constr_uh_e`	double	no
J_{sbx}^{e}	`constr_Jsbx`	double, **SPUME**	no
J_{sg}^{e}	`constr_Jsg_e`	double, **SPUME**	no
J_{sbx}^{e}	`constr_Jsh_e`	double, **SPUME**	no

A.5 External links

A table sheet with additional info is found here:
https://docs.google.com/spreadsheets/d/1rVRycLnCyaWJLwnV47u30Vokp7vRu68og3OhlDbSjDU/edit?usp=sharing

A.6 Model

A model instance is created using `ocp_model = acados_ocp_model()`. It contains all model definitions for simulation and for usage in the OCP solver. See table A.11 for the available options. Furthermore, see `ocp_model.model_struct` to see what other fields can be set via direct access.

A.7 Solver & Options

An instance of the solver options class is created by using: `ocp_opts = acados_ocp_opts()`. Together with the model these options are used when instancing the solver interface class: `ocp = acados_ocp(ocp_model, ocp_opts)`.
Tables A.12, A.13 and A.14 show (almost) all available options. These options are set in Matlab via `ocp_opts.set(<stringid>, <value>)`. Furthermore, the struct `ocp_opts.opts_struct` can be used as a reference for what other fields are available.
Note that some options of the solver can be modified after creation using the routine: `set(<stringid>, <value>)`.
Some options can only be set before the solver is created, especially options that influence the memory requirements of OCP solver, such as the modules used in the formulation, the QP solver, etc.

Table A.11: Model `set(id, data)` options

String id	Data type	Description	Required
`name`	string	model name, used for code generation, default: `'ocp_model'`	no
`T`	double	end time	yes
`sym_x`	`CasADi` expr.	state vector x in problem formulation in sec. A.1	yes
`sym_u`	`CasADi` expr.	control vector u in problem formulation in sec. A.1	only in OCP
`sym_xdot`	`CasADi` expr.	derivative of the state $\dot{x}$ in implicit dynamics eq. (A.3)	if IRK is used
`sym_z`	`CasADi` expr.	algebraic state z in implicit dynamics eq. (A.3)	no, only with IRK
`sym_p`	`CasADi` expr.	parameters p of the problem formulation in sec. A.1	no

$\vdots$

Additionally, options from tables A.1, A.2, A.3, A.4, A.5, A.6, A.7, A.8, A.9 and A.10, apply here.

$\vdots$

Table A.12: Solver options

String identifier	Type	Default	Description
Code generation			
`compile_interface`	string	`'auto'`	in (`'auto'`, `'true'`, `'false'`)
`codgen_model`	string	`'true'`	in (`'true'`, `'false'`)
`compile_model`	string	`'true'`	in (`'true'`, `'false'`)
`output_dir`	string	`'build'`	codegen output directory
Shooting nodes			
`param_scheme_N`	int > 1	10	uniform grid: number of shooting nodes; acts together with end time `T` from model.
`shooting_nodes` or `param_-scheme_shooting_nodes`	doubles	`[]`	nonuniform grid option 1: direct definition of the shooting node times
`time_steps`	doubles	`[]`	nonuniform grid option 2: definition of deltas between shooting nodes
Integrator			
`sim_method`	string	`'irk'`	`'erk'`, `'irk'`, `'irk_gnsf'`
`sim_method_num_stages`	int	4	Runge-Kutta int. stages: (1) RK1, (2) RK2, (4) RK4
`sim_method_num_steps`	int	1	
`sim_method_newton_iter`	int	3	
`gnsf_detect_struct`	string	`'true'`	
NLP solver			
`nlp_solver`	string	`'sqp'`	in (`'sqp'`, `'sqp_rti'`)
`nlp_solver_max_iter`	int > 1	100	maximum number of NLP iterations
`nlp_solver_tol_stat`	double	10^{-6}	stopping criterion
`nlp_solver_tol_eq`	double	10^{-6}	stopping criterion
`nlp_solver_tol_ineq`	double	10^{-6}	stopping criterion
`nlp_solver_tol_comp`	double	10^{-6}	stopping criterion
`nlp_solver_ext_qp_res`	int	0	compute QP residuals at each NLP iteration
`nlp_solver_step_length`	double	1.0	fixed step length in SQP algorithm
`rti_phase`	int	0	RTI phase: (1) preparation, (2) feedback, (0) both
QP solver			
`qp_solver`	string	$\longrightarrow$	Defines the quadratic programming solver and condensing strategy. See table A.13
`qp_solver_iter_max`	int	50	maximum number of iterations per QP solver call
`qp_solver_cond_ric_alg`	int	0	factorize hessian in the condensing: (0) no, (1) yes
`qp_solver_ric_alg`	int	0	HPIPM specific
`qp_solver_warm_start`	int	0	(0) cold start, (1) warm start primal variables, (2) warm start and dual variables
`warm_start_first_qp`	int	0	warm start even in first SQP iteration: (0) no, (1) yes
globalization			
`globalization`	string	`'fixed_step'`	globalization strategy in (`'fixed_step'`, `'merit_backtracking'`), note `merit_backtracking` is a preliminary implementation.
`alpha_min`	double	0.05	minimum step-size, relevant for globalization
`alpha_reduction`	double	0.7	step-size reduction factor, relevant for globalization
Hessian approximation			
`nlp_solver_exact_hessian`	string	`'false'`	use exact hessian calculation: (”)in (`'true'`, `'false'`), use exact
`regularize_method`	string	$\longrightarrow$	Defines the hessian regularization method. See table A.14
`levenberg_marquardt`	double	0.0	in case of a singular hessian, setting this > 0 can help convergence
`exact_hess_dyn`	int	1	in (0, 1), compute and use hessian in dynamics, only if `'nlp_solver_-`

Table A.13: Solver `set('qp_solver', <stringid>)` options. The availability depends on for which solver interfaces acados was linked to.

Solver lib	Condensing	String identifier
HPIPM	partial	`partial_condensing_hpipm`*
	full	`full_condensing_hpipm`
HPMPC	partial	`partial_condensing_hpmpc`
OOQP	partial	`partial_condensing_ooqp`
	full	`full_condensing_ooqp`
OSQP	partial	`partial_condensing_osqp`
QORE	full	`full_condensing_qore`
qpDUNES	partial	`partial_condensing_qpdunes`
qpOASES	full	`full_condensing_qpoases`

* default

Table A.14: Solver `set('regularize_method', <stringid>)` options

String identifier	Description
`no_regularize`*	dont regularize
`mirror`	see Verschueren2017
`project`	see Verschueren2017
`project_reduc_hess`	preliminary
`convexify`	see Verschueren2017

* default