1 Problem Formulation

acados can handle continuous-time optimal control problems (OCP) of the following form:

$$
\begin{aligned}
&\texttt{/* Cost function, see section 3 */}\\
\min_{x(\cdot),u(\cdot),z(\cdot),s(\cdot),s^{\mathrm{e}}} \quad & \int_0^T l(x(\tau),u(\tau),z(\tau),t,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}t +\\
& m(x(T),z(T),p)+\frac{1}{2}\begin{bmatrix} s_{\mathrm{l}}^{\mathrm{e}}\\ s_{\mathrm{u}}^{\mathrm{e}}\\ 1\end{bmatrix}^{\top}\begin{bmatrix} Z_{\mathrm{l}}^{\mathrm{e}} & 0 & z_{\mathrm{l}}^{\mathrm{e}}\\ 0 & Z_{\mathrm{u}}^{\mathrm{e}} & z_{\mathrm{u}}^{\mathrm{e}}\\ z_{\mathrm{l}}^{\mathrm{e}\top} & z_{\mathrm{u}}^{\mathrm{e}\top} & 0\end{bmatrix}\begin{bmatrix} s_{\mathrm{l}}^{\mathrm{e}}\\ s_{\mathrm{u}}^{\mathrm{e}}\\ 1\end{bmatrix} && (1)\\
&\texttt{/* Initial values, see section 4.1 */}\\
\text{s.t.} \quad & \underline{x}_0 \le J_{\mathrm{bx},0}\,x(0) \le \overline{x}_0, && (2)\\
&\texttt{/* Nonlinear constraints on the initial shooting node */}\\
& \underline{h}^0 \le h^0(x(0),u(0),z(0),p)+J_{\mathrm{sh}}^0 s_{\mathrm{l,h}}^0, && (3)\\
& h^0(x(0),u(0),z(0),p)-J_{\mathrm{sh}}^0 s_{\mathrm{u,h}}^0 \le \overline{h}^0, && (4)\\
&\texttt{/* Dynamics, see section 2 */}\\
& f_{\mathrm{impl}}(x(t),\dot{x}(t),u(t),z(t),t,p)=0, && t\in[0,T), \quad (5)\\
&\texttt{/* Path constraints with lower bounds, see section 4.2 */}\\
& \underline{h} \le h(x(t),u(t),z(t),p)+J_{\mathrm{sh}}\,s_{\mathrm{l,h}}(t), && t\in(0,T), \quad (6)\\
& \underline{x} \le J_{\mathrm{bx}}\,x(t)+J_{\mathrm{sbx}}\,s_{\mathrm{l,bx}}(t), && t\in(0,T), \quad (7)\\
& \underline{u} \le J_{\mathrm{bu}}\,u(t)+J_{\mathrm{sbu}}\,s_{\mathrm{l,bu}}(t), && t\in[0,T), \quad (8)\\
& \underline{g} \le C\,x(t)+D\,u(t)+J_{\mathrm{sg}}\,s_{\mathrm{l,g}}(t), && t\in[0,T), \quad (9)\\
& 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 (10)\\
& s_{\mathrm{l,h}}^0 \ge 0, && (11)\\
&\texttt{/* Path constraints with upper bounds, see section 4.2 */}\\
& h(x(t),u(t),z(t),p)-J_{\mathrm{sh}}\,s_{\mathrm{u,h}}(t) \le \overline{h}, && t\in(0,T), \quad (12)\\
& J_{\mathrm{bx}}x(t)-J_{\mathrm{sbx}}\,s_{\mathrm{u,bx}}(t) \le \overline{x}, && t\in(0,T), \quad (13)\\
& J_{\mathrm{bu}}u(t)-J_{\mathrm{sbu}}\,s_{\mathrm{u,bu}}(t) \le \overline{u}, && t\in[0,T), \quad (14)\\
& Cx(t)+Du(t)-J_{\mathrm{sg}}\,s_{\mathrm{u,g}} \le \overline{g}, && t\in[0,T), \quad (15)\\
& 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), \quad (16)\\
& s_{\mathrm{u,h}}^0 \ge 0, && (17)\\
&\texttt{/* Terminal constraints with lower bounds, see section 4.3 */}\\
& \underline{h}^{\mathrm{e}} \le h^{\mathrm{e}}(x(T),p)+J_{\mathrm{sh}}^{\mathrm{e}}\,s_{\mathrm{l,h}}^{\mathrm{e}}, && (18)\\
& \underline{x}^{\mathrm{e}} \le J_{\mathrm{bx}}^{\mathrm{e}}\,x(T)+J_{\mathrm{sbx}}^{\mathrm{e}}\,s_{\mathrm{l,bx}}^{\mathrm{e}}, && (19)\\
& \underline{g}^{\mathrm{e}} \le C^{\mathrm{e}}\,x(T)+J_{\mathrm{sg}}^{\mathrm{e}}\,s_{\mathrm{l,g}}^{\mathrm{e}} \le \overline{g}^{\mathrm{e}}, && (20)\\
& s_{\mathrm{l,h}}^{\mathrm{e}},s_{\mathrm{l,bx}}^{\mathrm{e}},s_{\mathrm{l,bu}}^{\mathrm{e}},s_{\mathrm{l,g}}^{\mathrm{e}}\ge 0, && (21)\\
&\texttt{/* Terminal constraints with upper bound, see section 4.3 */}\\
& h^{\mathrm{e}}(x(T),p)-J_{\mathrm{sh}}^{\mathrm{e}}\,s_{\mathrm{u,h}}^{\mathrm{e}} \le \overline{h}^{\mathrm{e}}, && (22)\\
& J_{\mathrm{bx}}^{\mathrm{e}}\,x(T)-J_{\mathrm{sbx}}^{\mathrm{e}}\,s_{\mathrm{u,bx}}^{\mathrm{e}} \le \overline{x}^{e}, && (23)\\
& C^{\mathrm{e}}\,x(T)-J_{\mathrm{sg}}^{\mathrm{e}}\,s_{\mathrm{u,g}}^{\mathrm{e}} \le \overline{g}^{\mathrm{e}} && (24)\\
& s_{\mathrm{u,h}}^{\mathrm{e}},s_{\mathrm{u,bx}}^{\mathrm{e}},s_{\mathrm{u,bu}}^{\mathrm{e}},s_{\mathrm{u,g}}^{\mathrm{e}}\ge 0, && (25)
\end{aligned}
$$

with

- state vector $x : \mathbb{R} \to \mathbb{R}^{n_x}$
- control vector $u : \mathbb{R} \to \mathbb{R}^{n_u}$
- algebraic state vector $z : \mathbb{R} \to \mathbb{R}^{n_z}$
- model parameters $p \in \mathbb{R}^{n_p}$
- slacks for initial constraints $s^0_{u,h} \in \mathbb{R}^{n^0_s}$ and $s^0_{l,h} \in \mathbb{R}^{n^0_s}$
- slacks for path constraints $s_l(t) = (s_{l,bu}, s_{l,bx}, s_{l,g}, s_{l,h}) \in \mathbb{R}^{n_s}$ and $s_u(t) = (s_{u,bu}, s_{u,bx}, s_{u,g}, s_{u,h}) \in \mathbb{R}^{n_s}$
- slacks for terminal constraints $s^e_l(t) = (s^e_{l,bx}, s^e_{l,g}, s^e_{l,h}) \in \mathbb{R}^{n^e_s}$ and $s^e_u(t) = (s^e_{u,bx}, s^e_{u,g}, s^e_{u,h}) \in \mathbb{R}^{n^e_s}$

While the user specifies the OCP in continuous time, the problem is internally discretized using the **multiple shooting** approach and a piecewise constant control parametrization. In particular, the algebraic variables returned by acados are associated with the beginning of a shooting interval. Thus, there are no algebraic variables available at the terminal node.

Document Purpose. This document describes the class of optimal control problems that can be implemented in acados via its MATLAB and PYTHON interfaces. This documentation is (most likely) not exhaustive and does not contain a full description of all options provided by the PYTHON and MATLAB interfaces.

Abbreviations. 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

2 Dynamics

The system dynamics equation (5) is replaced with a discrete-time dynamic system. The dynamics can be formulated in different ways in acados: As implicit equations in continuous time (26), or as explicit equations in continuous time (27) or directly as discrete-time dynamics (28). This section and Table 1 summarize these options.

2.1 Implicit Dynamics

The most general way to provide a continuous-time ODE in acados is to define the function $f_{\text{impl}} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_z} \times \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x+n_z}$ which is fully implicit DAE formulation describing the system as:

$$f_{\text{impl}}(x, \dot{x}, u, z, t, p) = 0. \tag{26}$$

acados can discretize f_{impl} with a classical implicit Runge-Kutta (`'IRK'`), a structure exploiting implicit Runge-Kutta method (`'GNSF'`) [4], or a lifted-Newton direct collocation method (`'LIFTED_IRK'`) [8]. These discretization methods are set via `AcadosOcpOptions.integrator_type`. The implicit dynamics f_{impl} need to be provided as a `CasADi` expression and set as `AcadosModel.f_impl_expr`.

Note. Time-varying dynamics are at the moment only available via the PYTHON interface and only in combination with the classical implicit Runge-Kutta (`'IRK'`) integrator.

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

$$f_{\text{expl}}(x, u, t, p) = \dot{x}. \tag{27}$$

The explicit dynamics f_{expl} need to be provided as a `CasADi` expression and set as `AcadosModel.f_expl_expr`. The explicit integrator is used if `AcadosOcpOptions.integrator_type` is set to `'ERK'`.

Note. Time-varying dynamics are at the moment only available via the PYTHON interface and only in combination with the classical implicit Runge-Kutta (`'IRK'`) integrator.

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

$$x_{i+1} = f_{\text{discrete}}(x_i, u_i, p_i). \tag{28}$$

The explicit dynamics f_{discrete} need to be provided as a `CasADi` expression and set as `AcadosModel.disc_dyn_expr`. The discrete dynamics are used if `AcadosOcpOptions.integrator_type` is set to `'DISCRETE'`.

Term	Attribute/Property	Data type	Required
x	`AcadosModel.x`	`CasADi` synmbolic	yes
$\dot{x}$	`AcadosModel.xdot`	`CasADi` synmbolic	yes for `'IRK'`, `'LIFTED_IRK'`, `'GNSF'`
u	`AcadosModel.u`	`CasADi` synmbolic	no
z	`AcadosModel.z`	`CasADi` synmbolic	no
p	`AcadosModel.p`	`CasADi` synmbolic	no
t	`AcadosModel.t`	`CasADi` synmbolic	no
integrator type	`AcadosOcpOptions.integrator_type`, `AcadosSimOptions.integrator_type`	str in [`'ERK'`, `'IRK'`, `'DISCRETE'`, `'LIFTED_IRK'`, `'GNSF'`]	yes
f_{impl}	`AcadosModel.f_impl_expr`	`CasADi` expression	yes for `'IRK'`, `'LIFTED_IRK'`, `'GNSF'`
f_{expl}	`AcadosModel.f_expl_expr`	`CasADi` expression	yes for `'ERK'`
f_{disc}	`AcadosModel.disc_dyn_expr`	`CasADi` expression	yes for `'DISCRETE'`

Table 1: Dynamics definitions.

3 Cost

The objective in (1) comprises

- $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_z} \times \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}$, which is the Lagrange cost term, and
- $m : \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}$, which is the Mayer cost term.

There are different acados modules that provide different possibilities to describe these cost terms. The cost module used for l and m is determined from `AcadosOcpCost.cost_type_0`, `AcadosOcpCost.cost_type` and `AcadosOcpCost.cost_type_e`, respectively.

Setting the slack penalties in equation (1) is done in the same way for all cost modules, see Table 2 for an overview.

Slack penalties for the initial node can be set through the appropriate fields `cost_xx_0`.

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, `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 $s_l(t)$, $s_l^e(t)$, $s_u(t)$ and $s_u^e(t)$ are determined by acados from the associated matrices (Z_l, Z_u, J_{sh}, J_{sg}, J_{sbu}, J_{sbx} etc.).

By default, the Lagrange cost term provided in continuous time is internally integrated using the explicit Euler method, `AcadosOcpOptions.cost_discretization = 'EULER'`, which allows for a seamless OCP discretization with a nonuniform time grid. This means that all cost terms, except for the terminal one, are weighted with the corresponding time step. If the time steps are $\Delta t_0, \ldots, \Delta t_{N-1}$, the total cost is given by $c_{\text{total}} = \Delta t_0 \cdot l(x_0, u_0, p_0, z_0) + \cdots + \Delta t_{N-1} \cdot l(x_{N-1}, u_{N-1}, p_{N-1}, z_{N-1}) + m(x_N, p_N)$.

If a nonlinear least-squares or convex-over-nonlinear cost is used, the cost can also be integrated using the same integration scheme, which is used for the dynamics. This is obtained for `AcadosOcpOptions.cost_discretization = 'INTEGRATOR'`.

Term	Attribute/Property	Data type	Required
Z_l^0	`AcadosOcpCost.Zl_0`	double, **DIAG**	no
Z_u^0	`AcadosOcpCost.Zu_0`	double, **DIAG**	no
z_l^0	`AcadosOcpCost.zl_0`	double	no
z_u^0	`AcadosOcpCost.zu_0`	double	no
Z_l	`AcadosOcpCost.Zl`	double, **DIAG**	no
Z_u	`AcadosOcpCost.Zu`	double, **DIAG**	no
z_l	`AcadosOcpCost.zl`	double	no
z_u	`AcadosOcpCost.zu`	double	no
Z_l^e	`AcadosOcpCost.Zl_e`	double, **DIAG**	no
Z_u^e	`AcadosOcpCost.Zu_e`	double, **DIAG**	no
z_l^e	`AcadosOcpCost.zl_e`	double	no
z_u^e	`AcadosOcpCost.zu_e`	double	no

Table 2: Penalties on slack variable

3.1 Cost module `'AUTO'`

Set `AcadosOcpCost.cost_type_0`, `AcadosOcpCost.cost_type`, `AcadosOcpCost.cost_type_e` to `'AUTO'` in order to use this cost module. 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 3 shows the available options.

Note. The cost type `'AUTO'` is only available from the MATLAB interface.

Term	Attribute/Property	Data type	Required
l	`AcadosModel.cost_expr_ext_cost_0`	`CasADi` expression	no
l	`AcadosModel.cost_expr_ext_cost`	`CasADi` expression	yes
m	`AcadosModel.cost_expr_ext_cost_e`	`CasADi` expression	no

Table 3: Cost module `'AUTO'` options

3.2 Cost module `EXTERNAL`: External cost/Economic cost

Set `AcadosOcpCost.cost_type_0`, `AcadosOcpCost.cost_type`, `AcadosOcpCost.cost_type_e` to `ext_cost`. See Table 4 for the available options.

Term	Attribute/Property	Data type	Required
l	`AcadosModel.cost_expr_ext_cost`	`CasADi` expression	yes
m	`AcadosModel.cost_expr_ext_cost_e`	`CasADi` expression	yes

Table 4: Cost module `external` options

3.3 Cost module `'LINEAR_LS'`: Linear least-squares cost

In order to activate the `linear least squares` cost module, set `AcadosOcpCost.cost_type_0`, `AcadosOcpCost.cost_type`, `AcadosOcpCost.cost_type_e` to `'LINEAR_LS'`.

The Lagrange cost term has the form

$$l(x,u,z) = \frac{1}{2}\left\| \underbrace{V_x\, x + V_u\, u + V_z\, z}_{y} - y_{\text{ref}} \right\|_W^2 \tag{29}$$

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 weighting matrix. The vector $y_{\text{ref}} \in \mathbb{R}^{n_y}$ is the reference.

Similarly, the Mayer cost term has the form

$$m(x,u,z) = \frac{1}{2}\left\| \underbrace{V_x^{\text{e}} x}_{y^{\text{e}}} - y_{\text{ref}}^{\text{e}} \right\|_{W^{\text{e}}}^2 \tag{30}$$

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 weighting matrix. The vector $y_{\text{ref}}^{\text{e}} \in \mathbb{R}^{n_{y^{\text{e}}}}$ is the reference.

Additionally, a different cost for the initial node can be set using the same form as (29) and the appropriate fields. See Table 5 for the available options of this cost module.

Term	Attribute/Property	Data type	Required
V_x^0	`AcadosOcpCost.Vx_0`	double	no
V_u^0	`AcadosOcpCost.Vu_0`	double	no
V_z^0	`AcadosOcpCost.Vz_0`	double	no
W^0	`AcadosOcpCost.W_0`	double	no
y_{ref}^0	`AcadosOcpCost.yref_0`	double	no
V_x	`AcadosOcpCost.Vx`	double	yes
V_u	`AcadosOcpCost.Vu`	double	yes
V_z	`AcadosOcpCost.Vz`	double	yes
W	`AcadosOcpCost.W`	double	yes
y_{ref}	`AcadosOcpCost.yref`	double	yes
V_x^{e}	`AcadosOcpCost.Vx_e`	double	yes
W^{e}	`AcadosOcpCost.W_e`	double	yes
$y_{\text{ref}}^{\text{e}}$	`AcadosOcpCost.yref_e`	double	yes

Table 5: Cost module `'LINEAR_LS'` options

3.4 Cost module `'NONLINEAR_LS'`: Nonlinear least-squares cost

In order to activate the `nonlinear least squares` cost module, set `AcadosOcpCost.cost_type_0`, `AcadosOcpCost.cost_type`, `AcadosOcpCost.cost_type_e` to `'NONLINEAR_LS'`.

The nonlinear least-squares cost has the same basic form as eqns. (29 - 30) of the `'LINEAR_LS'` cost module. The only difference is that y and y^{e} are defined in terms of `CasADi` expressions, instead of via matrices V_x, V_u, V_z and V_x^{e}. The same note about the initial node applies to this cost module as well. See Table 6 for the available options of this cost module.

3.5 Cost module `CONVEX_OVER_NONLINEAR`: Convex-over-nonlinear cost

If the cost takes the form

$$l(x,u,t,p) = \psi(r(x,u,z,t,p) - y_{\text{ref}}, t, p), \qquad m(x,p) = \psi^{\text{e}}(r^{\text{e}}(x,t,p) - y_{\text{ref}}^{\text{e}}, p), \tag{31}$$

with ψ and ψ^{e} convex, we say the cost has convex-over-nonlinear structure. In this case, a Generalized Gauss-Newton Hessian might be used for the SQP subproblems [7], set `AcadosOcpOptions.hessian_approx = 'GAUSS_NEWTON'`.

Term	Attribute/Property	Data type	Required
y^0	`AcadosModel.cost_y_expr_0`	`CasADi` expression	no
W^0	`AcadosOcpCost.W_0`	double	no
y^0_{ref}	`AcadosOcpCost.yref_0`	double	no
y	`AcadosModel.cost_y_expr`	`CasADi` expression	yes
W	`AcadosOcpCost.W`	double	yes
y_{ref}	`AcadosOcpCost.yref`	double	yes
y^{e}	`AcadosModel.cost_y_expr_e`	`CasADi` expression	yes
W^{e}	`AcadosOcpCost.W_e`	double	yes
$y^{\text{e}}_{\text{ref}}$	`AcadosOcpCost.yref_e`	double	yes

Table 6: Cost module `'NONLINEAR_LS'` options

See Table 7 for the available options of this cost module.

Note. Convex-over-nonlinear costs are at the moment only available via the PYTHON interface.

Term	Attribute/Property	Data type	Required
y^0	`AcadosModel.cost_y_expr_0`	`CasADi` expression	no
y^0_{ref}	`AcadosOcpCost.yref_0`	double	no
r^0	`AcadosModel.cost_r_in_psi_expr_0`	`CasADi` symbolic	no
ψ^0	`AcadosModel.cost_psi_expr_0`	`CasADi` expression	no
y	`AcadosModel.cost_y_expr`	`CasADi` expression	yes
y_{ref}	`AcadosOcpCost.yref`	double	yes
r	`AcadosModel.cost_r_in_psi_expr`	`CasADi` symbolic	no
ψ	`AcadosModel.cost_psi_expr`	`CasADi` expression	no
y^{e}	`AcadosModel.cost_y_expr_e`	`CasADi` expression	yes
$y^{\text{e}}_{\text{ref}}$	`AcadosOcpCost.yref_e`	double	yes
r^{e}	`AcadosModel.cost_r_in_psi_expr_e`	`CasADi` symbolic	yes
ψ^{e}	`AcadosModel.cost_psi_expr_e`	`CasADi` expression	yes

Table 7: Cost module `'CONVEX_OVER_NONLINEAR'` options

4 Constraints

This section is about how to define the constraints equations (2 - 25).

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.

Additionally, bounds on u and general linear constraints are also enforced on the initial node by default. On the other hand, bounds on x and nonlinear constraints are fully split and have to be explicitly stated with `_0` correspondence to be enforced on the initial node.

4.1 Initial State

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

Two possibilities exist to define the initial state constraint (2): a simple syntax and an extended syntax.

Simple syntax. Via the simple syntax the full initial state is defined, $x(0) = \overline{x}_0$. The corresponding options are found in Table 8.

Term	Attribute/Property	Data type	Required
$\overline{x}_0$	`AcadosConstraints.x0`	double	no

Table 8: Simple syntax for setting the initial state

Extended syntax. The extended syntax allows to define upper and lower bounds on a subset of states. The options for the extended syntax are found in Table 9. Here, $\mathcal{I}_{\text{bx},0}$ is a vector of length n_x containing only zeros and ones.

Term	Attribute/Property	Data type	Required
$\overline{x}_0$	`AcadosConstraints.ubx_0`	double	no
$\underline{x}_0$	`AcadosConstraints.lbx_0`	double	no
$J_{\text{bx},0} = \text{diag}(\mathcal{I}_{\text{bx},0})$	`AcadosConstraints.idxbx_0`	double	no

Table 9: Extended syntax for setting the initial state

4.2 Path Constraints

Table 10 shows the options for defining the path constraints equations (3 - 17). The matrices $J_\star$ are translated into arrays of integers `idx`$\star$, see PYTHONdocumentation. These matrices are described as follows:

- J_{sh} maps lower slack vectors $s_{\text{l,h}}(t)$ and upper slack vectors $s_{\text{u,h}}(t)$ onto the nonlinear constraint expressions $h(x,u,p)$.
- J_{bx}, J_{bu} map $x(t)$ and $u(t)$ onto their bounds vectors $\underline{x}$, $\overline{x}$ and $\underline{u}$, $\overline{u}$, respectively.
- J_{sx}, J_{su} map lower slack vectors $s_{\text{l,bx}}(t)$, $s_{\text{l,bu}}(t)$ and upper slack vectors $s_{\text{u,bx}}(t)$, $s_{\text{u,bu}}(t)$ onto $x(t)$ and $u(t)$, respectively.
- J_{sg} maps lower slack vectors $s_{\text{l,g}}(t)$ and upper slack vectors $s_{\text{u,g}}(t)$ onto lower and upper equality bounds $\underline{g}$, $\overline{g}$, respectively.
- C, D map $x(t)$ and $u(t)$ onto lower and upper inequality bounds $\underline{g}$, $\overline{g}$ (polytopic constraints).
- J_{sh}^0 maps lower slack vectors $s_{\text{l,h}}^0$ and upper slack vectors $s_{\text{u,h}}^0$ onto the nonlinear initial constraint expressions $h^0(x(0),u(0),p)$.

4.3 Terminal Constraints

Table 11 shows the options for defining the terminal constraints equations (18 - 25). Here, matrices

- J_{sh}^{e} maps lower slack vectors $s_{\text{l,h}}^{\text{e}}(t)$ and upper slack vectors $s_{\text{u,h}}^{\text{e}}(t)$ onto nonlinear terminal constraint expressions $h^{\text{e}}(x(T),p)$.
- J_{bx}^{e} maps $x(T)$ onto its bounds vectors $\underline{x}^{\text{e}}$ and $\overline{x}^{\text{e}}$.
- $J_{\text{sbx}}^{\text{e}}$ maps lower slack vectors $s_{\text{l,bx}}^{\text{e}}$ and upper slack vectors $s_{\text{u,bx}}^{\text{e}}$ onto $x(T)$.
- J_{sg}^{e} maps lower slack vectors $s_{\text{l,g}}^{\text{e}}(t)$ and upper slack vectors $s_{\text{u,g}}^{\text{e}}(t)$ onto lower and upper equality bounds $\underline{g}^{\text{e}}$, $\overline{g}^{\text{e}}$, respectively.
- C^{e} maps $x(T)$ onto lower and upper inequality bounds $\underline{g}^{\text{e}}$, $\overline{g}^{\text{e}}$ (polytopic constraints).

Term	Attribute/Property	Data type	Required
J_{bx}	`AcadosConstraints.idxbx`	double	no
$\overline{x}$	`AcadosConstraints.ubx`	double	no
$\underline{x}$	`AcadosConstraints.lbx`	double	no
J_{bu}	`AcadosConstraints.idxbu`	double	no
$\overline{u}$	`AcadosConstraints.ubu`	double	no
$\underline{u}$	`AcadosConstraints.lbu`	double	no
C	`AcadosConstraints.C`	double	no
D	`AcadosConstraints.D`	double	no
$\overline{g}$	`AcadosConstraints.ug`	double	no
$\underline{g}$	`AcadosConstraints.lg`	double	no
h^0	`AcadosModel.con_h_expr_0`	`CasADi` expression	no
$\overline{h}^0$	`AcadosConstraints.uh_0`	double	no
$\underline{h}^0$	`AcadosConstraints.lh_0`	double	no
h	`AcadosModel.con_h_expr`	`CasADi` expression	no
$\overline{h}$	`AcadosConstraints.uh`	double	no
$\underline{h}$	`AcadosConstraints.lh`	double	no
$J_{\text{sbx}} = \text{diag}(\mathcal{I}_{\text{sbx}})$	`AcadosConstraints.idxsbx`	double	no
$J_{\text{sbu}} = \text{diag}(\mathcal{I}_{\text{sbu}})$	`AcadosConstraints.idxsbu`	double	no
$J_{\text{sg}} = \text{diag}(\mathcal{I}_{\text{sg}})$	`AcadosConstraints.idxsg`	double	no
$J_{\text{sh}} = \text{diag}(\mathcal{I}_{\text{sh}})$	`AcadosConstraints.idxsh`	double	no
$J^0_{\text{sh}} = \text{diag}(\mathcal{I}^0_{\text{sh}})$	`AcadosConstraints.idxsh_0`	double	no

Table 10: Path constraints options

Term	Attribute/Property	Data type	Required
$J^{\text{e}}_{\text{bx}} = \text{diag}(\mathcal{I}^{\text{e}}_{\text{bx}})$	`AcadosConstraints.idxbx_e`	double	no
$\overline{x}^{\text{e}}$	`AcadosConstraints.ubx_e`	double	no
$\underline{x}^{\text{e}}$	`AcadosConstraints.lbx_e`	double	no
C^{e}	`AcadosConstraints.C_e`	double	no
$\overline{g}^{\text{e}}$	`AcadosConstraints.ug_e`	double	no
$\underline{g}^{\text{e}}$	`AcadosConstraints.lg_e`	double	no
h^{e}	`AcadosModel.con_h_expr_e`	`CasADi` expression	no
$\overline{h}^{\text{e}}$	`AcadosConstraints.uh_e`	double	no
$\underline{h}^{\text{e}}$	`AcadosConstraints.lh_e`	double	no
$J^{\text{e}}_{\text{sbx}} = \text{diag}(\mathcal{I}^{\text{e}}_{\text{sbx}})$	`AcadosConstraints.idxsbx_e`	double	no
$J^{\text{e}}_{\text{sg}} = \text{diag}(\mathcal{I}^{\text{e}}_{\text{sg}})$	`AcadosConstraints.idxsg_e`	double	no
$J^{\text{e}}_{\text{sh}} = \text{diag}(\mathcal{I}^{\text{e}}_{\text{sh}})$	`AcadosConstraints.idxsh_e`	double	no

Table 11: Terminal constraints options

5 Solver & Options

Tables 12, 13 and 5 show some of the available options. A full list of the available options is given on the docs page.

Note that some options of the solver can be modified after creation using the routine. Some options can only be set before the solver is created, especially options that influence the memory requirements of the OCP solver, such as the modules used in the formulation, the QP solver, etc.

Attribute/Property	Type	Default	Description
Code generation			
`compile_interface`	string	`'auto'`	in (`'auto'`, `'true'`, `'false'`)
Shooting nodes			
`tf`	float		prediction horizon
`N_horizon`	int > 1		uniform grid: number of shooting nodes; acts together with end time/length of prediction horizon `tf`.
`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			
`integrator_type`	string	`'ERK'`	`'ERK'`, `'IRK'`, `'LIFTED_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	
`sim_method_detect_gnsf`	string	`'true'`	
NLP solver			
`nlp_solver_type`	string	`'SQP'`	in (`'SQP'`, `'SQP_RTI'`„ ”DDP”)
`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
`nlp_solver_warm_start_first_qp`	bool	false	warm start even in first SQP iteration
QP solver			
`qp_solver`	string	$\longrightarrow$	Defines the quadratic programming solver and condensing strategy. See Table 13
`qp_solver_iter_max`	int	50	maximum number of iterations per QP solver call
`qp_solver_cond_N`	int	N	new horizon after partial condensing, set to `param_scheme_N` by default
`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
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			
`hessian_approx`	string	`'GAUSS_NEWTON'`	use exact Hessian or Gauss-Newton approximation, (`'EXACT'`, `'GAUSS_NEWTON'`)
`regularize_method`	string	$\longrightarrow$	Defines the hessian regularization method. See Table 5
`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_exact_hessian' = 'true'`
`exact_hess_cost`	int	1	in (0, 1), only if `'nlp_solver_exact_hessian' = 'true'`
`exact_hess_constr`	int	1	in (0, 1), only if `'nlp_solver_exact_hessian' = 'true'`
Other			
`print_level`	int ≥ 0	0	verbosity of the solver: (0) silent, (> 0) print first QP problems and solution during SQP

Table 12: Solver options that can be set as attributes/properties of `AcadosOcpOptions`.

Solver lib	Condensing	Attribute/Property	Reference
HPIPM	partial	PARTIAL_CONDENSING_HPIPM*	[6]
	full	full_condensing_HPIPM	[6]
HPMPC	partial	PARTIAL_CONDENSING_HPMPC	[5]
OSQP	partial	PARTIAL_CONDENSING_OSQP	[9]
qpDUNES	partial	PARTIAL_CONDENSING_QPDUNES	[3]
qpOASES	full	full_condensing_QPOASES	[2]
DAQP	full	full_condensing_DAQP	[1]

* default

Table 13: QP solver options. Note that you might need to recompile acados wit the corresponding flags in order to use the above QP solvers.

Attribute/Property	Description
NO_REGULARIZE*	don't regularize
MIRROR	compare [10]
PROJECT	compare [10]
PROJECT_REDUC_HESS	preliminary
CONVEXIFY	compare [10], does not work in combination with nonlinear constraints

* default

Table 14: Regularization options.

References

[1] D. Arnstrom, A. Bemporad, and D. Axehill. A dual active-set solver for embedded quadratic programming using recursive LDL^T updates. *IEEE Transactions on Automatic Control*, 2022. doi: 10.1109/TAC.2022.3176430.

[2] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl. qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[3] J. V. Frasch, S. Sager, and M. Diehl. A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computations*, 7(3):289–329, 2015.

[4] J. Frey, R. Quirynen, D. Kouzoupis, G. Frison, J. Geisler, A. Schild, and M. Diehl. Detecting and exploiting Generalized Nonlinear Static Feedback structures in DAE systems for MPC. In *Proceedings of the European Control Conference (ECC)*, 2019.

[5] G. Frison. *Algorithms and Methods for High-Performance Model Predictive Control*. PhD thesis, Technical University of Denmark (DTU), 2015.

[6] G. Frison and M. Diehl. HPIPM: a high-performance quadratic programming framework for model predictive control. In *Proceedings of the IFAC World Congress*, Berlin, Germany, July 2020.

[7] F. Messerer, K. Baumgärtner, and M. Diehl. Survey of sequential convex programming and generalized Gauss-Newton methods. *ESAIM: Proceedings and Surveys*, 71:64–88, 2021.

[8] R. Quirynen, S. Gros, and M. Diehl. Lifted implicit integrators for direct optimal control. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 3212–3217, 2015.

[9] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020. doi: 10.1007/s12532-020-00179-2. URL `https://doi.org/10.1007/s12532-020-00179-2`.

[10] R. Verschueren, M. Zanon, R. Quirynen, and M. Diehl. A sparsity preserving convexification procedure for indefinite quadratic programs arising in direct optimal control. *SIAM Journal of Optimization*, 27(3):2085–2109, 2017.