1 Problem Formulation

In acados, one can formulate continuous-time optimal control problems (OCP) of the following form using the class AcadosOcp:

```
/* Cost function, see section 3 */
```

$$\min_{x(\cdot),u(\cdot),z(\cdot),s(\cdot),s^e} \int_0^T l(x(t),u(t),z(t),t,p(t),p_{\text{global}}) + \frac{1}{2} \begin{bmatrix} s_l(t) \\ s_u(t) \\ 1 \end{bmatrix}^\top \begin{bmatrix} Z_l & 0 & z_l \\ 0 & Z_u & z_u \\ z_l^\top & z_u^\top & 0 \end{bmatrix} \begin{bmatrix} s_l(t) \\ s_u(t) \\ 1 \end{bmatrix} dt +$$

$$m(x(T),z(T),p(T),p_{\text{global}}) + \frac{1}{2} \begin{bmatrix} s_l^e \\ s_u^e \\ 1 \end{bmatrix}^\top \begin{bmatrix} Z_l^e & 0 & z_l^e \\ 0 & Z_u^e & z_u^e \\ z_l^{e\top} & z_u^{e\top} & 0 \end{bmatrix} \begin{bmatrix} s_l^e \\ s_u^e \\ 1 \end{bmatrix} \tag{1}$$

```
/* Initial values, see section 4.1 */
```

s.t.
$$\underline{x}_0 \le J_{\text{bx}}^0 x(0) \le \overline{x}_0, \tag{2}$$

```
/* Nonlinear constraints on the initial shooting node */
```

$$\underline{h}^0 \le h^0(x(0),u(0),z(0),p(0),p_{\text{global}}) + J_{\text{sh}}^0 s_{l,h}^0, \tag{3}$$

$$h^0(x(0),u(0),z(0),p(0),p_{\text{global}}) - J_{\text{sh}}^0 s_{u,h}^0 \le \overline{h}^0, \tag{4}$$

```
/* Dynamics, see section 2 */
```

$$f_{\text{impl}}(x(t),\dot{x}(t),u(t),z(t),t,p(t),p_{\text{global}}) = 0, \qquad t \in [0,T), \tag{5}$$

```
/* Path constraints with lower bounds, see section 4.2 */
```

$$\underline{h} \le h(x(t),u(t),z(t),p(t),p_{\text{global}}) + J_{\text{sh}} s_{l,h}(t), \qquad t \in (0,T), \tag{6}$$

$$\underline{x} \le J_{\text{bx}} x(t) + J_{\text{sbx}} s_{l,\text{bx}}(t), \qquad t \in (0,T), \tag{7}$$

$$\underline{u} \le J_{\text{bu}} u(t) + J_{\text{sbu}} s_{l,\text{bu}}(t), \qquad t \in [0,T), \tag{8}$$

$$\underline{g} \le C x(t) + D u(t) + J_{\text{sg}} s_{l,g}(t), \qquad t \in [0,T), \tag{9}$$

$$s_{l,h}(t), s_{l,\text{bx}}(t), s_{l,\text{bu}}(t), s_{l,g}(t) \ge 0, \qquad t \in [0,T), \tag{10}$$

$$s_{l,h}^0 \ge 0, \tag{11}$$

```
/* Path constraints with upper bounds, see section 4.2 */
```

$$h(x(t),u(t),z(t),p(t),p_{\text{global}}) - J_{\text{sh}} s_{u,h}(t) \le \overline{h}, \qquad t \in (0,T), \tag{12}$$

$$J_{\text{bx}} x(t) - J_{\text{sbx}} s_{u,\text{bx}}(t) \le \overline{x}, \qquad t \in (0,T), \tag{13}$$

$$J_{\text{bu}} u(t) - J_{\text{sbu}} s_{u,\text{bu}}(t) \le \overline{u}, \qquad t \in [0,T), \tag{14}$$

$$C x(t) + D u(t) - J_{\text{sg}} s_{u,g} \le \overline{g}, \qquad t \in [0,T), \tag{15}$$

$$s_{u,h}(t), s_{u,\text{bx}}(t), s_{u,\text{bu}}(t), s_{u,g}(t) \ge 0, \qquad t \in [0,T), \tag{16}$$

$$s_{u,h}^0 \ge 0, \tag{17}$$

```
/* Terminal constraints with lower bounds, see section 4.2 */
```

$$\underline{h}^e \le h^e(x(T),p(T),p_{\text{global}}) + J_{\text{sh}}^e s_{l,h}^e, \tag{18}$$

$$\underline{x}^e \le J_{\text{bx}}^e x(T) + J_{\text{sbx}}^e s_{l,\text{bx}}^e, \tag{19}$$

$$\underline{g}^e \le C^e x(T) + J_{\text{sg}}^e s_{l,g}^e \le \overline{g}^e, \tag{20}$$

$$s_{l,h}^e, s_{l,\text{bx}}^e, s_{l,\text{bu}}^e, s_{l,g}^e \ge 0, \tag{21}$$

```
/* Terminal constraints with upper bounds, see section 4.2 */
```

$$h^e(x(T),p(T),p_{\text{global}}) - J_{\text{sh}}^e s_{u,h}^e \le \overline{h}^e, \tag{22}$$

$$J_{\text{bx}}^e x(T) - J_{\text{sbx}}^e s_{u,\text{bx}}^e \le \overline{x}^e, \tag{23}$$

$$C^e x(T) - J_{\text{sg}}^e s_{u,g}^e \le \overline{g}^e \tag{24}$$

$$s_{u,h}^e, s_{u,\text{bx}}^e, s_{u,\text{bu}}^e, s_{u,g}^e \ge 0, \tag{25}$$

with

- state vector $x : \mathbb{R} \to \mathbb{R}^{n_x}$
- control vector $u : \mathbb{R} \to \mathbb{R}^{n_u}$
- algebraic state vector $z : \mathbb{R} \to \mathbb{R}^{n_z}$
- time-varying problem parameters $p : \mathbb{R} \to \in \mathbb{R}^{n_p}$
- global problem parameters $p_{\text{global}} \in \mathbb{R}^{n_{p_{\text{global}}}}$
- slacks for initial constraints $s_{u,h}^0 \in \mathbb{R}^{n_s^0}$ and $s_{l,h}^0 \in \mathbb{R}^{n_s^0}$
- 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 $s_l^e(t) = (s_{l,bx}^e, s_{l,g}^e, s_{l,h}^e) \in \mathbb{R}^{n_s^e}$ and $s_u^e(t) = (s_{u,bx}^e, s_{u,g}^e, s_{u,h}^e) \in \mathbb{R}^{n_s^e}$
- selection matrices $J_\star, J_{s\star}$, i.e. they are all zeros except for a single one entry in each row.
- diagonal matrices Z_l, Z_u, Z_l^e, Z_u^e

While the user specifies the OCP in continuous time, the problem is internally discretized using the **multiple shooting** approach and a piecewise constant parametrization for the control and parameter trajectory. 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 formulated and solved in acados via the class AcadosOcp, available in the 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.

In particular, this document does not cover the acados problem formulation and interface for multi-phase OCP [5].

2 Dynamics

The system dynamics equation (5) is internally replaced with a discrete-time dynamic system. The dynamics can be formulated in different ways in acados: As implicit equations in continuous time (26), 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:

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

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') [9]. 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

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

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 symbolic	yes for `'IRK'`, `'LIFTED_IRK'`, `'GNSF'`
u	`AcadosModel.u`	CasADi symbolic	no
z	`AcadosModel.z`	CasADi symbolic	no
p	`AcadosModel.p`	CasADi symbolic	no
p_{global}	`AcadosModel.p_global`	CasADi symbolic	no
t	`AcadosModel.t`	CasADi symbolic	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. Note that the matrices Z_l^*, Z_u^* are diagonal matrices and provided to acados by only specifying their diagonal values.

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 vector	no
Z_u^0	`AcadosOcpCost.Zu_0`	double vector	no
z_l^0	`AcadosOcpCost.zl_0`	double vector	no
z_u^0	`AcadosOcpCost.zu_0`	double vector	no
Z_l	`AcadosOcpCost.Zl`	double vector	no
Z_u	`AcadosOcpCost.Zu`	double vector	no
z_l	`AcadosOcpCost.zl`	double vector	no
z_u	`AcadosOcpCost.zu`	double vector	no
Z_l^e	`AcadosOcpCost.Zl_e`	double vector	no
Z_u^e	`AcadosOcpCost.Zu_e`	double vector	no
z_l^e	`AcadosOcpCost.zl_e`	double vector	no
z_u^e	`AcadosOcpCost.zu_e`	double vector	no

Table 2: Penalties on slack variable. Note that the matrices Z_l^*, Z_u^* are diagonal matrices and provided to acados by only specifying their diagonal values.

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. Table 3 shows the available options.

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

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

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

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

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 matrix	no
V_u^0	AcadosOcpCost.Vu_0	double matrix	no
V_z^0	AcadosOcpCost.Vz_0	double matrix	no
W^0	AcadosOcpCost.W_0	double matrix, positive semidefinite	no
y_{ref}^0	AcadosOcpCost.yref_0	double vector	no
V_x	AcadosOcpCost.Vx	double matrix	yes
V_u	AcadosOcpCost.Vu	double matrix	yes
V_z	AcadosOcpCost.Vz	double matrix	yes
W	AcadosOcpCost.W	double matrix, positive semidefinite	yes
y_{ref}	AcadosOcpCost.yref	double vector	yes
V_x^{e}	AcadosOcpCost.Vx_e	double matrix	yes
W^{e}	AcadosOcpCost.W_e	double matrix, positive semidefinite	yes
$y_{\text{ref}}^{\text{e}}$	AcadosOcpCost.yref_e	double vector	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

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

with $\psi(r)$ and $\psi^{\text{e}}(r^{\text{e}})$ 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 [8], which is used whenever AcadosOcpOptions.hessian_approx = 'GAUSS_NEWTON'. 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
W^0	`AcadosOcpCost.W_0`	double matrix, positive semidefinite	no
y_ref^0	`AcadosOcpCost.yref_0`	double vector	no
y	`AcadosModel.cost_y_expr`	CasADi expression	yes
W	`AcadosOcpCost.W`	double matrix, positive semidefinite	yes
y_ref	`AcadosOcpCost.yref`	double vector	yes
y^e	`AcadosModel.cost_y_expr_e`	CasADi expression	yes
W^e	`AcadosOcpCost.W_e`	double matrix, positive semidefinite	yes
y_ref^e	`AcadosOcpCost.yref_e`	double vector	yes

Table 6: Cost module 'NONLINEAR_LS' options.

Term	Attribute/Property	Data type	Required
y^0	`AcadosModel.cost_y_expr_0`	CasADi expression	no
y_ref^0	`AcadosOcpCost.yref_0`	double vector	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 vector	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_ref^e	`AcadosOcpCost.yref_e`	double vector	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. Convex-over-nonlinear costs are at the moment only available via the PYTHON interface.

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.

There is preliminary support for one-sided constraints for the QP solvers HPIPM and DAQP by setting lower/upper bounds to ACADOS_INFTY.

4.1 Initial State Constraint

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 vector	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. Note that the matrix J_{bx}^0 is all zeros except for a single entry with the value one in each row. The matrix is internally computed from a vector of integers specifying the 0-based column indices of the 1 entries in J_{bx}^0. This vector is provided to acados via the attribute/property idxbx_0.

As an example, consider idxbx_0 = $[0, 1, 3]$ and assume $n_x = 5$, i.e. the zeroth, first and third component of $x_0 \in \mathbb{R}^5$ are constrained. In this case, the corresponding matrix J_{bx}^0 would be given by

$$J_{\mathrm{bx}}^0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{32}$$

Term	Attribute/Property	Data type	Required
$\overline{x}_0$	AcadosConstraints.ubx_0	double	no
$\underline{x}_0$	AcadosConstraints.lbx_0	double	no
J_{bx}^0	AcadosConstraints.idxbx_0	integer vector	no

Table 9: Extended syntax for setting the initial state.

4.2 Path Constraints and Terminal Constraints

Table 10 and 11 show the options for defining path constraints, equations (3) - (17), and terminal constraints, equations (18) - (25), respectively.

Note that the matrices $J_\star$ are all zeros except for a single entry with the value one in each row. The matrices are computed from vectors of integers, provided to acados via the attributes/properties idx$\star$, specifying the 0-based column indices of the 1 entries in $J_\star$. We refer to Section 4.1 for an example.

The matrices are described as follows:

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

- 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_{sbx}, J_{sbu} 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} maps lower slack vectors $s_{\mathrm{l,g}}(t)$ and upper slack vectors $s_{\mathrm{u,g}}(t)$ onto lower and upper equality bounds $\underline{g}$, $\overline{g}$, 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 $s_{\mathrm{l,h}}^0$ and upper slack vectors $s_{\mathrm{u,h}}^0$ onto the nonlinear initial constraint expressions $h^0(x(0), u(0), p, p_{\mathrm{global}})$.

Similarly for the terminal constraints:

- $J_{\mathrm{sh}}^{\mathrm{e}}$ maps lower slack vectors $s_{\mathrm{l,h}}^{\mathrm{e}}(t)$ and upper slack vectors $s_{\mathrm{u,h}}^{\mathrm{e}}(t)$ onto nonlinear terminal constraint expressions $h^{\mathrm{e}}(x(T), p, p_{\mathrm{global}})$.

- $J_{\mathrm{bx}}^{\mathrm{e}}$ maps $x(T)$ onto its bounds vectors $\underline{x}^{\mathrm{e}}$ and $\overline{x}^{\mathrm{e}}$.

- $J_{\mathrm{sbx}}^{\mathrm{e}}$ maps lower slack vectors $s_{\mathrm{l,bx}}^{\mathrm{e}}$ and upper slack vectors $s_{\mathrm{u,bx}}^{\mathrm{e}}$ 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`	integer vector	no
$\overline{x}$	`AcadosConstraints.ubx`	double vector	no
$\underline{x}$	`AcadosConstraints.lbx`	double vector	no
J_{bu}	`AcadosConstraints.idxbu`	integer vector	no
$\overline{u}$	`AcadosConstraints.ubu`	double vector	no
$\underline{u}$	`AcadosConstraints.lbu`	double vector	no
C	`AcadosConstraints.C`	double matrix	no
D	`AcadosConstraints.D`	double matrix	no
$\overline{g}$	`AcadosConstraints.ug`	double vector	no
$\underline{g}$	`AcadosConstraints.lg`	double vector	no
h^0	`AcadosModel.con_h_expr_0`	CasADi expression	no
$\overline{h}^0$	`AcadosConstraints.uh_0`	double vector	no
$\underline{h}^0$	`AcadosConstraints.lh_0`	double vector	no
h	`AcadosModel.con_h_expr`	CasADi expression	no
$\overline{h}$	`AcadosConstraints.uh`	double vector	no
$\underline{h}$	`AcadosConstraints.lh`	double vector	no
J_{sbx}	`AcadosConstraints.idxsbx`	integer vector	no
J_{sbu}	`AcadosConstraints.idxsbu`	integer vector	no
J_{sg}	`AcadosConstraints.idxsg`	integer vector	no
J_{sh}	`AcadosConstraints.idxsh`	integer vector	no
J_{sh}^0	`AcadosConstraints.idxsh_0`	integer vector	no

Table 10: Path constraints options. Note that 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.

Term	Attribute/Property	Data type	Required
J_{bx}^{e}	`AcadosConstraints.idxbx_e`	integer vector	no
$\overline{x}^{\text{e}}$	`AcadosConstraints.ubx_e`	double vector	no
$\underline{x}^{\text{e}}$	`AcadosConstraints.lbx_e`	double vector	no
C^{e}	`AcadosConstraints.C_e`	double matrix	no
$\overline{g}^{\text{e}}$	`AcadosConstraints.ug_e`	double vector	no
$\underline{g}^{\text{e}}$	`AcadosConstraints.lg_e`	double vector	no
h^{e}	`AcadosModel.con_h_expr_e`	CasADi expression	no
$\overline{h}^{\text{e}}$	`AcadosConstraints.uh_e`	double vector	no
$\underline{h}^{\text{e}}$	`AcadosConstraints.lh_e`	double vector	no
$J_{\text{sbx}}^{\text{e}}$	`AcadosConstraints.idxsbx_e`	integer vector	no
J_{sg}^{e}	`AcadosConstraints.idxsg_e`	integer vector	no
J_{sh}^{e}	`AcadosConstraints.idxsh_e`	integer vector	no

Table 11: Terminal constraints options.

5   Solver Options

Tables 12, 13 and 14 show some of the available options. This list is however not up to date. We recommend to check the docs page generated from the docstrings in PYTHON, as well as the definition of the class AcadosOcpOptions available both in MATLAB and PYTHON, for a full list of the available options.

Note that some options of the solver can be modified after its creation. 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'), MATLAB only!
Shooting nodes			
tf	double		prediction horizon (denoted with T in this document)
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'	in ('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 the 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_horizon	new horizon after partial condensing; no condensing 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 both primal and dual variables
Globalization			
globalization	string	'FIXED_STEP'	globalization strategy in ('FIXED_STEP', 'MERIT_BACKTRACKING'), note that 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 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_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*	[7]
	full	FULL_CONDENSING_HPIPM	[7]
HPMPC	partial	PARTIAL_CONDENSING_HPMPC	[6]
OSQP	partial	PARTIAL_CONDENSING_OSQP	[10]
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 with the corresponding flags in order to use the above QP solvers.

Attribute/Property	Description
NO_REGULARIZE*	don't regularize
MIRROR	compare [11]
PROJECT	compare [11]
PROJECT_REDUC_HESS	preliminary
CONVEXIFY	compare [11], 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] J. Frey, K. Baumgärtner, G. Frison, and M. Diehl. Multi-phase optimal control problems for efficient nonlinear model predictive control with acados. *arXiv preprint arXiv:2408.07382*, 2024.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.