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 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 + \\
& && \quad 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.} && \underline{x}_0 \le J_{\mathrm{bx},0}\, x(0) \le \bar{x}_0, && (2) \\
& && \texttt{/* Nonlinear constraints on initial shooting node */} \\
& && \underline{h}^0 \le h^0(x(0),u(0),p) + J_{\mathrm{sh}}^0\, s_{\mathrm{l,h}}^0, && (3) \\
& && h^0(x(0),u(0),p) - J_{\mathrm{sh}}^0\, s_{\mathrm{u,h}}^0 \le \bar{h}^0, && (4) \\
& && \texttt{/* Dynamics, see section 2 */} \\
& && f_{\mathrm{impl}}(x(t),\dot{x}(t),u(t),z(t),p) = 0, \quad t \in [0,\, T), && (5) \\
& && \texttt{/* Path constraints with lower bounds, see section 4.2 */} \\
& && \underline{h} \le h(x(t),u(t),p) + J_{\mathrm{sh}}\, s_{\mathrm{l,h}}(t), \quad t \in (0,\, T), && (6) \\
& && \underline{x} \le J_{\mathrm{bx}}\, x(t) + J_{\mathrm{sbx}}\, s_{\mathrm{l,bx}}(t), \quad t \in (0,\, T), && (7) \\
& && \underline{u} \le J_{\mathrm{bu}}\, u(t) + J_{\mathrm{sbu}}\, s_{\mathrm{l,bu}}(t), \quad t \in [0,\, T), && (8) \\
& && \underline{g} \le C\, x(t) + D\, u(t) + J_{\mathrm{sg}}\, s_{\mathrm{l,g}}(t), \quad t \in [0,\, T), && (9) \\
& && s_{\mathrm{l,h}}(t), s_{\mathrm{l,bx}}(t), s_{\mathrm{l,bu}}(t), s_{\mathrm{l,g}}(t) \ge 0, \quad t \in [0,\, T), && (10) \\
& && s_{\mathrm{l,h}}^0 \ge 0, && (11) \\
& && \texttt{/* Path constraints with upper bounds, see section 4.2 */} \\
& && h(x(t),u(t),p) - J_{\mathrm{sh}}\, s_{\mathrm{u,h}}(t) \le \bar{h}, \quad t \in (0,\, T), && (12) \\
& && J_{\mathrm{bx}} x(t) - J_{\mathrm{sbx}}\, s_{\mathrm{u,bx}}(t) \le \bar{x}, \quad t \in (0,\, T), && (13) \\
& && J_{\mathrm{bu}} u(t) - J_{\mathrm{sbu}}\, s_{\mathrm{u,bu}}(t) \le \bar{u}, \quad t \in [0,\, T), && (14) \\
& && C x(t) + D u(t) - J_{\mathrm{sg}}\, s_{\mathrm{u,g}} \le \bar{g}, \quad t \in [0,\, T), && (15) \\
& && s_{\mathrm{u,h}}(t), s_{\mathrm{u,bx}}(t), s_{\mathrm{u,bu}}(t), s_{\mathrm{u,g}}(t) \ge 0, \quad t \in [0,\, T), && (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 \bar{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 \bar{h}^{\mathrm{e}}, && (22) \\
& && J_{\mathrm{bx}}^{\mathrm{e}}\, x(T) - J_{\mathrm{sbx}}^{\mathrm{e}}\, s_{\mathrm{u,bx}}^{\mathrm{e}} \le \bar{x}^{e}, && (23) \\
& && C^{\mathrm{e}}\, x(T) - J_{\mathrm{sg}}^{\mathrm{e}}\, s_{\mathrm{u,g}}^{\mathrm{e}} \le \bar{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 $x : \mathbb{R} \to \mathbb{R}^{n_\mathrm{x}}$
- 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 initial constraints $s^0_{\mathrm{u,h}} \in \mathbb{R}^{n^0_\mathrm{s}}$ and $s^0_{\mathrm{l,h}} \in \mathbb{R}^{n^0_\mathrm{s}}$
- 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 $s^\mathrm{e}_\mathrm{l}(t) = (s^\mathrm{e}_{\mathrm{l,bx}}, s^\mathrm{e}_{\mathrm{l,g}}, s^\mathrm{e}_{\mathrm{l,h}}) \in \mathbb{R}^{n^\mathrm{e}_\mathrm{s}}$ and $s^\mathrm{e}_\mathrm{u}(t) = (s^\mathrm{e}_{\mathrm{u,bx}}, s^\mathrm{e}_{\mathrm{u,g}}, s^\mathrm{e}_{\mathrm{u,h}}) \in \mathbb{R}^{n^\mathrm{e}_\mathrm{s}}$

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 in 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.

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 (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 summarizes the options.

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:

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

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.

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

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

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

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

Table 1: Dynamics definitions

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

3 Cost

There are different acados modules to model the cost functions in equation (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 (1) is done in the same way for all cost modules, see table 2 for an overview. Moreover,

Table 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

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 $s_\mathrm{l}(t)$, $s_\mathrm{l}^\mathrm{e}(t)$, $s_\mathrm{u}(t)$ 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.).

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 3 shows the available options.

Table 3: Cost module `auto` options

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

3.2 Cost module: `external`

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

Table 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

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

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

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

Similarly, the Mayer cost term has the form

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

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

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

Table 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

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. (29 - 30) 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 6 for the available options of this cost module.

4 Constraints

This section is about how to define the constraints equations (2) and (3 - 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.

Table 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
$y^{\mathrm{e}}_{\mathrm{ref}}$	cost_y_ref_e	double	yes

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 (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 7.

Table 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 8.

Table 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
$J_{\mathrm{bx},0}$	constr_Jbx_0	double	no

4.2 Path Constraints

Table 9 shows the options for defining the path constraints equations (3 - 17). The matrices $J_\star$ are translated into arrays of integers idx⋆, see Python documentation. These matrices 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 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)
- J^0_{sh} maps lower slack vectors $s^0_{\mathrm{l,h}}$ and upper slack vectors $s^0_{\mathrm{u,h}}$ onto the non-linear initial constraint expressions $h^0(x(0), u(0), p)$.

4.3 Terminal Constraints

Table 10 shows the options for defining the terminal constraints equations (18 - 25). Here, matrices

Table 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^0	`constr_expr_h_0`	`CasADi` expression	no
$\underline{h}^0$	`constr_lh_0`	double	no
$\bar{h}^0$	`constr_uh_0`	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_{sh}	`constr_Jsh`	double, **SPUME**	no

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

5 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 11 for the available options. Furthermore, see `ocp_model.model_struct` or `acados_ocp_model.m` to see what other fields can be set via direct access.

6 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 12, 13 and 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` and `acados_ocp_opts.m` 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 10: Terminal constraints options

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

Table 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. 1	yes
`sym_u`	`CasADi` expr.	control vector u in problem formulation in sec. 1	only in OCP
`sym_xdot`	`CasADi` expr.	derivative of the state $\dot{x}$ in implicit dynamics eq. (5)	if IRK is used
`sym_z`	`CasADi` expr.	algebraic state z in implicit dynamics eq. (5)	no, only with IRK
`sym_p`	`CasADi` expr.	parameters p of the problem formulation in sec. 1	no
		⋮	
		Additionally, options from tables 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10, apply here.	
		⋮	

Table 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 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
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 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 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
OSQP	partial	partial_condensing_osqp
qpDUNES	partial	partial_condensing_qpdunes
qpOASES	full	full_condensing_qpoases
DAQP	full	full_condensing_daqp

* default

Table 14: Solver set(’regularize_method’, <stringid>) options

String identifier	Description
no_regularize*	don‘t regularize
mirror	see Verschueren2017
project	see Verschueren2017
project_reduc_hess	preliminary
convexify	see Verschueren2017

* default