1 Problem Formulation

`acados` can handle the following optimization problem

$$
\begin{aligned}
&&& \texttt{/* Cost function, see section 3 */} \\
&\min_{x(\cdot),\, u(\cdot),\, z(\cdot),\, s(\cdot),\, s^{\mathrm{e}}} && \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{/* Dynamics, see section 2 */} \\
&&& f_{\mathrm{impl}}(x(t), \dot{x}(t), u(t), z(t), p) = 0, \qquad t \in [0,\, T), && (3) \\
&&& \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), \qquad t \in [0,\, T), && (4) \\
&&& \underline{x} \le J_{\mathrm{bx}}\, x(t) + J_{\mathrm{sbx}}\, s_{\mathrm{l,bx}}(t), \qquad t \in (0,\, T), && (5) \\
&&& \underline{u} \le J_{\mathrm{bu}}\, u(t) + J_{\mathrm{sbu}}\, s_{\mathrm{l,bu}}(t), \qquad t \in [0,\, T), && (6) \\
&&& \underline{g} \le C\, x(t) + D\, u(t) + J_{\mathrm{sg}}\, s_{\mathrm{l,g}}(t), \qquad t \in [0,\, T), && (7) \\
&&& s_{\mathrm{l,h}}(t), s_{\mathrm{l,bx}}(t), s_{\mathrm{l,bu}}(t), s_{\mathrm{l,g}}(t) \ge 0, \qquad t \in [0,\, T), && (8) \\
&&& \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}, \qquad t \in [0,\, T), && (9) \\
&&& J_{\mathrm{bx}} x(t) - J_{\mathrm{sbx}}\, s_{\mathrm{u,bx}}(t) \le \bar{x}, \qquad t \in (0,\, T), && (10) \\
&&& J_{\mathrm{bu}} u(t) - J_{\mathrm{sbu}}\, s_{\mathrm{u,bu}}(t) \le \bar{u}, \qquad t \in [0,\, T), && (11) \\
&&& C x(t) + D u(t) - J_{\mathrm{sg}}\, s_{\mathrm{u,g}} \le \bar{g}, \qquad t \in [0,\, T), && (12) \\
&&& s_{\mathrm{u,h}}(t), s_{\mathrm{u,bx}}(t), s_{\mathrm{u,bu}}(t), s_{\mathrm{u,g}}(t) \ge 0, \qquad t \in [0,\, T), && (13) \\
&&& \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}}, && (14) \\
&&& \underline{x}^{\mathrm{e}} \le J_{\mathrm{bx}}^{\mathrm{e}}\, x(T) + J_{\mathrm{sbx}}\, s_{\mathrm{l,bx}}^{\mathrm{e}}, && (15) \\
&&& \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}}, && (16) \\
&&& 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, && (17) \\
&&& \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}}, && (18) \\
&&& J_{\mathrm{bx}}^{\mathrm{e}}\, x(T) - J_{\mathrm{sbx}}\, s_{\mathrm{u,bx}}^{\mathrm{e}} \le \bar{x}^{\mathrm{e}}, && (19) \\
&&& C^{\mathrm{e}}\, x(T) - J_{\mathrm{sg}}^{\mathrm{e}}\, s_{\mathrm{u,g}}^{\mathrm{e}} \le \bar{g}^{\mathrm{e}} && (20) \\
&&& 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, && (21)
\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 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 also found in the PYTHON interface (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 under the `<ACADOS>/examples/acados_matlab_octave` directory. The MATLAB source code of `CasADi` is found here: `<ACADOS>/interfaces/acados_matlab_octave` and should serve as a more extensive 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 (3) is a placeholder for equations (22), (23) or (24). Therefore, the dynamics can be formulated in different ways in `acados`. 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:

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

`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.

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

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

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

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

Table 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

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.

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

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 you don't have to set slack variables $s_\mathrm{l}(t)$, $s_\mathrm{l}^\mathrm{e}(t)$, $s_\mathrm{u}(t)$ and $s_\mathrm{u}^\mathrm{e}(t)$ manually. They are part of the solution and its dimensions are determined by `acados` from the associated matrices (Z_l, Z_u, J_sh, J_sbx, J_sbu, 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 transcribe 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 form 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

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

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 $y_{\mathrm{ref}} \in \mathbb{R}^{n_y}$ is the reference.

Similarly, the Mayer cost term has the form

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

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

3.4 Cost module: `non-linear least squares`

In order to activate the `non-linear least squares` cost module, set `cost_type` to `nonlinear_ls`.

The `non-linear least squares` cost function has the same basic form as eqns. (25 - 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 6 for the available options of this cost module.

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
$y_{\mathrm{ref}}^{\mathrm{e}}$	`cost_y_ref_e`	double	yes

4 Constraints

This section is about how to define the constraints equations (2) and (4 - 21). The constraint type can be set using the identifier `'constr_type'` and `'constr_type_e'` for the path constraints and terminal constraints, respectively. The string identifier options are found in table 7. The default setting is `'bgh'`.

Table 7: Constraint type string identifier.

String identifier	Supported constraints: simple bounds	polytopic constr.	general non-linear constr.	positive definite constr.
`bgh`	yes	yes	yes	no
`bgp`	yes	yes	yes	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 8.

Table 8: 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 9.

Table 9: 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 10 shows the options for defining the path constraints equations (4 - 13). Here, matrices

- J_{sh}, maps lower slack vectors $s_{\mathrm{l,h}}(t)$ and upper slack vectors $s_{\mathrm{u,h}}(t)$ onto non-linear constraint expressions $h(x(t), u(t), p)$.
- 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 10: 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

4.3 Terminal Constraints

Table 11 shows the options for defining the terminal constraints equations (14 - 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_{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 11: 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
$J^{\mathrm{e}}_{\mathrm{sbx}}$	`constr_Jsh_e`	double, **SPUME**	no

Table 12: 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	?
`sym_u`	`CasADi` expr.	control vector u in problem formulation in sec. 1	?
`sym_xdot`	`CasADi` expr.	$\dot{x}$ in implicit dynamics eq. (3)	?
		Additionally, options from tables 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 and 11, apply here.	

5 External links

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

6 Model

A model instance is created using `ocp_model = acados_ocp_model()`. It contains all model definitions for either simulating the system or using it in the solver.

7 Solver

The solver options are created as an instance from the `acados_ocp_opts()` class.

Table 13: Solver `set(id, option)` 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 time `T` from model.
`shooting_nodes` or `param_scheme_shooting_nodes`	doubles	`[]`	nonuniform grid 1: direct definition of the shooting node times
`time_steps`	doubles	`[]`	nonuniform grid 2: definition of deltas between shooting nodes
NLP solver			
`nlp_solver`	string	`'sqp'`	in (`'sqp'`, `'sqp_rti'`)
`nlp_solver_exact_hessian`	string	`'false'`	use exact hessian calculation: (")in (`'true'`, `'false'`), use exact
`nlp_solver_max_iter`	int > 1	100	maximum number of NLP iterations
`nlp_solver_tol_stat`	double	10^{-6}	
`nlp_solver_tol_eq`	double	10^{-6}	
`nlp_solver_tol_ineq`	double	10^{-6}	
`nlp_solver_tol_comp`	double	10^{-6}	
`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 14
`globalization`	string	`'fixed_step'`	globalization
`alpha_min`	double	0.05	
`alpha_reduction`	double	0.7	
`qp_solver_iter_max`	int	50	
`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
`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'`	
`regularize_method`	string	$\longrightarrow$	Defines the hessian regularization method. See table 15
`print_level`	int ≥ 0	0	verbosity of the solver: (0) silent, (> 0) print solver output during solution finding
`levenberg_marquardt`	double	0.0	
`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'`

Table 14: Solver `set(id, option)` options, `'qp_solver'` options. The availability depends on for which solver interfaces `acados` was linked to.

Solver lib	Condensing	String identifier
HPIPM	partial	`partial_condensing_hpipm` (default)
	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`

Table 15: Solver `set(id, option)` options, `'regularize_method'` options.

String identifier	Description
`no_regularize`	don't regularize (default)
`mirror`	
`project`	
`project_reduc_hess`	
`convexify`	