SAC Library

Generated by Doxygen 1.8.11

Contents

Chapter 1

Todo List

Member sac::adjoint::adjoint (state_intp &x_intp, cost &J, Params &p)

Alex: make inputs const ref type.

Member sac::adjoint::operator() (const state_type &rho, state_type &rhodot, const double t)

Alex: decide if proj should go before lin.A()

Member sac::b_control::b_control (Params &p)

Alex: Make constructor explicit if it takes single arg.

Alex: Make constructor reference and pointer inputs const.

Member sac::cost::cost (state_intp &x_intp, Params &p)

Alex: Make constructor reference and pointer inputs const.

Member sac::cost::grad_mofx (vec_type &Gmofx)

: Alex: double check the order of proj and gproj calls

Member sac::get_min (InputIterator first, InputIterator last, Function &fn, T &min)

Alex: See about making inputs const type. Possibly also find max element since the entire list is searched.

Member sac::GoldenSection (T &fcost, Scalar a, Scalar b, Scalar c, Scalar &dt, Scalar &eps)

Alex: See about making inputs const type. Remove dt input parameter.

Member sac::inc_cost::grad (const double t, const vec_type &mx, vec_type &Glofx)

: Alex: is it ok that this needs to be called with proj(mx)?

Member sac::inc_cost::inc_cost (state_intp &x_intp, Params &p)

Alex: make inputs const ref type

Member sac::MinSearch (T &fcost, Scalar t0, Scalar tf, std::vector< Scalar > &lclMin, Scalar dt, Scalar eps)

Alex: See about making inputs const type.

Member sac::mode_insert_grad::mode_insert_grad (state_intp &x_intp, state_intp &rho_intp, u2_optimal &u2Opt, Params &p)

Alex: make inputs const ref type

Member sac::sgn (T val)

Alex: See about making input const type.

Member sac::simRho (adjoint &rho_dot, state_type &rho_Tf, double t0, double tf, std::vector< state_type > &rho_vec, std::vector< double > &rho_times)

Alex: See about making inputs const type.

Member sac::simX (sys_dynam &xdot, state_type &x0, double t0, double tf, std::vector< state_type > &x↩ _vec, std::vector< double > &times)

Alex: See about making inputs const type.

Member sac::state_intp::state_intp (std::vector< state_type > &states, std::vector< double > &times)

Alex: make inputs const ref type

Member sac::state_intp::state_intp (std::vector< state_type > &states, std::vector< double > &times, size_t xlength)

Alex: make inputs const ref type

Member sac::state_intp::update (std::vector< state_type > &states, std::vector< double > &times)

Alex: make inputs const ref type

Member sac::traj_cost::traj_cost (state_intp &rx_intp, std::vector< state_type > &ru2list, std::vector< state_type > &rTiTappTf, Params &p)

Alex: Make constructor reference and pointer inputs const.

Member sac::traj_cost::traj_cost (state_intp &rx_intp, std::vector< state_type > &ru2list, std::vector< state_type > &rTiTappTf, const vec_type &rmxdes_tf, Params &p)

Alex: Make constructor reference and pointer inputs const.

Member sac::u2_cost::u2_cost (u2_optimal &u2Opt, mode_insert_grad &dJdlam, Params &p)

Alex: see about converting inputs to const ref types.

Member sac::u2_optimal::operator() (const double t, state_type &u2Opt_curr)

Alex: incorporate u1

Member sac::u2_optimal::u2_optimal (state_intp &x_intp, state_intp &rho_intp, double &alpha, Params &p)

Alex: make inputs const ref type

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 sac Namespace Reference

Classes

- class adjoint
- class b_control
- class cost
- class inc_cost
- class inc_state_cost
- class mode_insert_grad
- class Params
- struct push_back_state_and_time
- class sac_step

  The class to carry out a sac control step.
- class state_intp
- class traj_cost
- class u2_cost
- class u2_optimal

Typedefs

- typedef std::vector< double > state_type
- typedef std::vector< double >::iterator iter_1d
- typedef std::vector< state_type >::iterator iter_2d
- typedef Eigen::MatrixXd mat_type
- typedef Eigen::MatrixXd vec_type

Functions

- template<class T >
  void State2Mat (const state_type &s, T &matOut)
- template<class T >
  void Mat2State (const T &mat, state_type &sOut)
- template<class Scalar >
  void AngleWrap (Scalar &theta)
- template<class T , class Scalar >
  void MinSearch (T &fcost, Scalar t0, Scalar tf, std::vector< Scalar > &lclMin, Scalar dt, Scalar eps)
- template<class T , class InputIterator , class Function >
  InputIterator get_min (InputIterator first, InputIterator last, Function &fn, T &min)
- size_t simX (sys_dynam &xdot, state_type &x0, double t0, double tf, std::vector< state_type > &x_vec, std↩
  ::vector< double > &times)
- size_t simRho (adjoint &rho_dot, state_type &rho_Tf, double t0, double tf, std::vector< state_type > &rho↩
  _vec, std::vector< double > &rho_times)
- template<typename T >
  int sgn (T val)
- template<class T , class Scalar >
  Scalar GoldenSection (T &fcost, Scalar a, Scalar b, Scalar c, Scalar &dt, Scalar &eps)
- template<class T >
  void SaveVec (const T &src, const char *outputFilename)

5.1.1 Typedef Documentation

5.1.1.1 typedef std::vector< double >::iterator sac::iter_1d

Definition at line 30 of file master.hpp.

5.1.1.2 typedef std::vector< state_type >::iterator sac::iter_2d

Definition at line 31 of file master.hpp.

5.1.1.3 typedef Eigen::MatrixXd sac::mat_type

Definition at line 33 of file master.hpp.

5.1.1.4 typedef std::vector< double > sac::state_type

Definition at line 29 of file master.hpp.

5.1.1.5 typedef Eigen::MatrixXd sac::vec_type

Definition at line 34 of file master.hpp.

5.1.2 Function Documentation

5.1.2.1 template<class Scalar > void sac::AngleWrap (Scalar & *theta*) [inline]

Replaces a scalar angle in radians with a wrapped version $\in [-\pi, \pi)$.

Parameters

in,out	*theta*	The angle in radians to be wrapped and returned.

Definition at line 347 of file master.hpp.

5.1.2.2 template<class T , class InputIterator , class Function > InputIterator sac::get_min (InputIterator *first,* InputIterator *last,* Function & *fn,* T & *min*)

Todo Alex: See about making inputs const type. Possibly also find max element since the entire list is searched.

Evaluates a callable exhaustively over each element of an iterable list domain to find the one that minimizes the callable.

Parameters

in	*first*	Iterator pointing to first element.
in	*last*	Iterator pointing to the end element.
in	*fn*	The callable to evaluate in search of a minimum.
out	*min*	The minimum value of the callable.

Returns

An iterator pointing to the iterable list object element that minimizes the callable.

Definition at line 266 of file master.hpp.

5.1.2.3 template<class T , class Scalar > Scalar sac::GoldenSection (T & *fcost,* Scalar *a,* Scalar *b,* Scalar *c,* Scalar & *dt,* Scalar & *eps*)

Todo Alex: See about making inputs const type. Remove dt input parameter.

Uses the golden section method to search for the minimum of a callable over a specified domain. It is assumed a single minimum exists over this domain.

Parameters

in	*fcost*	The callable to be searched to find a minimum.
in	*a,b,c*	Values of the callable at three domain points where $a < b < c$.
in	*dt*	Change in time.
in	*eps*	The desired tolerance in finding the minimum (eps<<1).

Returns

The scalar value of the minimum point between a and c.

Definition at line 195 of file master.hpp.

5.1.2.4 template<class T > void sac::Mat2State (const T & *mat,* state_type & *sOut*) [inline]

Converts a vector_type to a state_type.

Parameters

in	*mat*	A row/column vector.
out	*sOut*	A state_type vector with the same # of elements as the input matrix.

Definition at line 167 of file master.hpp.

5.1.2.5 template<class T , class Scalar > void sac::MinSearch (T & *fcost,* Scalar *t0,* Scalar *tf,* std::vector< Scalar > & *lclMin,* Scalar *dt,* Scalar *eps*)

Todo Alex: See about making inputs const type.

Samples a callable at specified intervals over a domain. Searches for zero-crossings. Uses golden section to search for minimizers on sub- intervals where zero crossings are detected. Returns vector of local minima.

Parameters

in	*fcost*	The callable to be searched to find local minima.
in	*t0,tf*	The domain on which to search for minima $t0 \leq t \leq tf$.
out	*lclMin*	An empty vector to store the local minima.
in	*dt*	Change in time.
in	*eps*	The desired tolerance in finding the minimum (eps<<1).

Definition at line 230 of file master.hpp.

5.1.2.6 template<class T > void sac::SaveVec (const T & *src,* const char * *outputFilename*)

Saves a 2D iterable object to a csv. To read in Matlab use: M = csvread('file.csv')

Parameters

in	*src*	The 2D iterable source object.
in	*outputFilename*	The filename to save the csv file as.

Definition at line 361 of file master.hpp.

5.1.2.7 template<typename T > int sac::sgn (T *val*)

Todo Alex: See about making input const type.

Computes the sign of a scalar.

Parameters

in	*val*	A state_type vector.

Returns

An integer -1, 0, or 1 depending on the sign of the input.

Definition at line 180 of file master.hpp.

5.1.2.8 size_t sac::simRho (adjoint & *rho_dot*, state_type & *rho_Tf*, double *t0*, double *tf*, std::vector< state_type > & *rho_vec*, std::vector< double > & *rho_times*)

Todo Alex: See about making inputs const type.

Simulates the adjoint / co-state, ρ, backwards in time from $\rho(t_f)$ to $\rho(t_0)$.

Parameters

in	*rho_dot*	The dynamics of the system co-state variable.
in,out	*rho_Tf*	The co-state variable at tf which gets integrated backwards to become the initial co-state variable.
in	*t0*	The initial time.
in	*tf*	The final time.
out	*rho_vec*	The vector of co-states resulting from integration.
out	*rho_times*	The vector of times resulting from integration.

Returns

The number of integration steps.

Definition at line 320 of file master.hpp.

5.1.2.9 size_t sac::simX (sys_dynam & *xdot*, state_type & *x0*, double *t0*, double *tf*, std::vector< state_type > & *x_vec*, std::vector< double > & *times*)

Todo Alex: See about making inputs const type.

Simulates the state forward in time from an initial state at t0 to the final state at tf.

Parameters

in	*xdot*	The dynamics of the system.
in,out	*x0*	The initial state which gets integrated to become the final state.
in	*t0*	The initial time.
in	*tf*	The final time.
out	*x_vec*	The vector of states resulting from integration.
out	*times*	The vector of times resulting from integration.

Returns

The number of integration steps.

Definition at line 292 of file master.hpp.

5.1.2.10 template<class T > void sac::State2Mat (const state_type & *s*, T & *matOut*) [inline]

Converts a state_type vector to a vector_type.

Parameters

in	*s*	A state_type vector with the same # of rows as the output.
out	*matOut*	A row/col vector.

Definition at line 154 of file master.hpp.

Chapter 6

Class Documentation

6.1 sac::adjoint Class Reference

`#include <adjoint.hpp>`

Public Member Functions

- adjoint (state_intp &x_intp, cost &J, Params &p)
- void operator() (const state_type &rho, state_type &rhodot, const double t)

Public Attributes

- state_intp & m_x_intp
- sys_lin m_lin
- inc_cost & m_lofx

6.1.1 Detailed Description

Warning

Class MUST BE MODIFIED BY USER to accomodate angle wrapping

Evaluates the rhs of $\dot{\rho}(t) = -\frac{\partial l}{\partial x}^T - \frac{\partial f}{\partial x}^T \rho(t)$ for integration of co-state variable, $\rho(t)$. Keeps references to state interpolator so that changes state trajectory are automatically accounted for.

Definition at line 13 of file adjoint.hpp.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 sac::adjoint::adjoint (state_intp & *x_intp*, cost & *J*, Params & *p*) [inline]

Todo Alex: make inputs const ref type.

Initializes references to user maintained state interpolation object and a cost object.

Parameters

in	*x_intp*	User maintained state interpolation object
in	*J*	The cost object required to provide incremental cost partial, $\frac{\partial l}{\partial x}$
in	*p*	SAC parameters

Definition at line 35 of file adjoint.hpp.

6.1.3 Member Function Documentation

6.1.3.1 void sac::adjoint::operator() (const state_type & *rho*, state_type & *rhodot*, const double *t*) [inline]

Returns the rhs of $\dot{\rho}(t) = -\frac{\partial l}{\partial x}^T - \frac{\partial f}{\partial x}^T \rho(t)$. The dynamics of co-state variable, $\rho(t)$.

Parameters

in	*rho*	The co-state variable at time t
out	*rhodot*	The dynamics of the co-state at time t
in	*t*	The time variable

Todo Alex: decide if proj should go before lin.A()

Definition at line 53 of file adjoint.hpp.

6.1.4 Member Data Documentation

6.1.4.1 sys_lin sac::adjoint::m_lin

Definition at line 23 of file adjoint.hpp.

6.1.4.2 inc_cost& sac::adjoint::m_lofx

Definition at line 24 of file adjoint.hpp.

6.1.4.3 state_intp& sac::adjoint::m_x_intp

Definition at line 22 of file adjoint.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/adjoint.hpp

6.2 sac::b_control Class Reference

`#include <b_control.hpp>`

Public Member Functions

- b_control (Params &p)
- void operator() (const double t, state_type &u_curr)
- void operator= (const state_type &u_switch)
- void no_saturate (const state_type &u_switch)
- void stimes (const double t1, const double t2)
- void clear ()

Public Attributes

- double m_tau1
- double m_tau2
- state_type m_u_switch

6.2.1 Detailed Description

This class stores switching control vectors. The control switches from a default, nominal control to the switching control signal when $\tau_1 \leq t \leq \tau_2$.

Definition at line 11 of file b_control.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 sac::b_control::b_control (Params & *p*) [inline]

Todo Alex: Make constructor explicit if it takes single arg.

Alex: Make constructor reference and pointer inputs const.

Constructor sets the nominal control to the zero vector.

Parameters

in	*p*	SAC parameters

Definition at line 24 of file b_control.hpp.

6.2.3 Member Function Documentation

6.2.3.1 void sac::b_control::clear ( ) [inline]

Re-set the switching control to the zero vector. Sets switching times τ_1 and τ_2 to 0.

Definition at line 80 of file b_control.hpp.

6.2.3.2 void sac::b_control::no_saturate (const state_type & *u_switch*) [inline]

Sets the value of the switching control when $\tau_1 \leq t \leq \tau_2$ without applying saturation.

Parameters

in	*u_switch*	The desired value of the switching control when $\tau_1 \leq t \leq \tau_2$.

Definition at line 66 of file b_control.hpp.

6.2.3.3 void sac::b_control::operator() (const double *t*, state_type & *u_curr*) [inline]

Returns the control at time t.

Parameters

in	*t*	The time at which to get the current control.
out	*u_curr*	The control vector at time t.

Definition at line 38 of file b_control.hpp.

6.2.3.4 void sac::b_control::operator= (const state_type & *u_switch*) [inline]

Sets the value of the switching control when $\tau_1 \leq t \leq \tau_2$. Also applies saturation to the vector.

Parameters

in	*u_switch*	The desired value of the switching control when $\tau_1 \leq t \leq \tau_2$.

Definition at line 50 of file b_control.hpp.

6.2.3.5 void sac::b_control::stimes (const double *t1*, const double *t2*) [inline]

Sets switching times τ_1 and τ_2 where the switching control is applied when $\tau_1 \leq t \leq \tau_2$.

Parameters

in	*t1,t2*	Desired values of switching times τ_1 and τ_2.

Definition at line 74 of file b_control.hpp.

6.2.4 Member Data Documentation

6.2.4.1 double sac::b_control::m_tau1

Definition at line 15 of file b_control.hpp.

6.2.4.2 double sac::b_control::m_tau2

Definition at line 15 of file b_control.hpp.

6.2.4.3 state_type sac::b_control::m_u_switch

Definition at line 16 of file b_control.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/b_control.hpp

6.3 sac::cost Class Reference

`#include <cost.hpp>`

Public Member Functions

- cost (state_intp &x_intp, Params &p)
- double get_term_cost ()
- void grad_mofx (vec_type &Gmofx)
- size_t compute_cost (state_type &term_cost)
- operator double ()
- size_t steps ()
- void update ()

Public Attributes

- inc_cost m_lofx
- state_intp & m_x_intp

6.3.1 Detailed Description

Class keeps track of the trajectory tracking cost. It stores the current state through a reference to a state interpolator object and thus only requires calling an update method to re-compute trajectory cost $J_1 = \int_{t_0}^{t_f} l(x(t))dt + m(x(t_f))$ after state / control updates.

Definition at line 12 of file cost.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 sac::cost::cost (state_intp & *x_intp*, Params & *p*) [inline]

Todo Alex: Make constructor reference and pointer inputs const.

Constructs a cost object from a state interpolation object and desired state trajectory.

Parameters

in	*x_intp*	state interpolation object
in	*p*	SAC parameters

Definition at line 31 of file cost.hpp.

6.3.3 Member Function Documentation

6.3.3.1 size_t sac::cost::compute_cost (state_type & *term_cost*)

The function computes the integral of $l(x)$ and appends it to a provided terminal cost to return cost $J_1 = \int_{t_0}^{t_f} l(x(t))dt + m(x(t_f))$.

Parameters

in,out	*term_cost*	the input terminal cost that gets updated with the total cost after integration.

Returns

The number of integration steps required.

Definition at line 106 of file cost.hpp.

6.3.3.2 double sac::cost::get_term_cost ( ) [inline]

Get the cost at the terminal time, $m(x(t_f))$.

Returns

$(x(t_f) - x_{des}(t_f))^T \; P_1 \; (x(t_f) - x_{des}(t_f))$, a quadratic form for $m(x(t_f))$.

Definition at line 42 of file cost.hpp.

6.3.3.3 void sac::cost::grad_mofx (vec_type & *Gmofx*) [inline]

Get the gradient of the terminal time, $D_x m(x(t_f))$.

Parameters

out	*Gmofx*	$P_1 \; (x(t_f) - x_{des}(t_f))$, which is $D_x m(x(t_f))^T$ assuming a quadratic form for the terimal cost.

Todo : Alex: double check the order of proj and gproj calls

Definition at line 57 of file cost.hpp.

6.3.3.4 sac::cost::operator double ( ) [inline]

Implicitly converts cost object to a double.

Returns

cost $J_1 = \int_{t_0}^{t_f} l(x(t))dt + m(x(t_f))$.

Definition at line 80 of file cost.hpp.

6.3.3.5 size_t sac::cost::steps ( ) [inline]

Returns

Returns # of integration steps in computing cost $J_1 = \int_{t_0}^{t_f} l(x(t))dt + m(x(t_f))$.

Definition at line 86 of file cost.hpp.

6.3.3.6 void sac::cost::update ( ) [inline]

Re-computes the cost stored in the cost object. This should be called to update the cost object after state / controls have been modified.

Definition at line 92 of file cost.hpp.

6.3.4 Member Data Documentation

6.3.4.1 inc_cost sac::cost::m_lofx

Definition at line 21 of file cost.hpp.

6.3.4.2 state_intp& sac::cost::m_x_intp

Definition at line 22 of file cost.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/cost.hpp

6.4 sac::inc_cost Class Reference

`#include <inc_cost.hpp>`

Public Member Functions

- inc_cost (state_intp &x_intp, Params &p)
- void operator() (const state_type &, state_type &dJdt, const double t)
- void grad (const double t, const vec_type &mx, vec_type &Glofx)
- double begin ()
- double end ()

Public Attributes

- state_intp & m_x_intp
- state_type m_x
- vec_type m_mxdes

6.4.1 Detailed Description

Warning

Projection and derivative of projection in testing

General incremental trajectory tracking cost, $l(x)$, for integration $J_1 = \int_{t_0}^{t_f} l(x)\,dt + m(x(t_f))$. Keeps references to state interpolator so that changes state trajectory are automatically accounted for.

Definition at line 13 of file inc_cost.hpp.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 sac::inc_cost::inc_cost (state_intp & *x_intp*, Params & *p*) [inline]

Todo Alex: make inputs const ref type

Initializes references to user maintained trajectory object and a pointer to the desired trajectory.

Parameters

in	*x_intp*	User maintained state interpolation object
in	*p*	SAC parameters

Definition at line 30 of file inc_cost.hpp.

6.4.3 Member Function Documentation

6.4.3.1 double sac::inc_cost::begin ( ) [inline]

Returns the initial time for integration, t_0

Returns

Initial integration time t_0

Definition at line 81 of file inc_cost.hpp.

6.4.3.2 double sac::inc_cost::end ( ) [inline]

Returns the final time for integration, t_f

Returns

Final integration time t_f

Definition at line 87 of file inc_cost.hpp.

6.4.3.3 void sac::inc_cost::grad (const double *t*, const vec_type & *mx*, vec_type & *Glofx*) [inline]

Todo : Alex: is it ok that this needs to be called with proj(mx)?

Computes the value of the gradient of the incremental cost, $D_x l(x)^T$.

Parameters

in	*t*	The current time
in	*mx*	The current state
out	*dldx*	The gradient $D_x l(x)^T$.

Definition at line 65 of file inc_cost.hpp.

6.4.3.4 void sac::inc_cost::operator() (const state_type & , state_type & *dJdt*, const double *t*) [inline]

Computes the value of incremental trajectory tracking cost, $l(x)$.

Parameters

in	*J*	The current cost
out	*dJdt*	The previous incremental cost
in	*t*	The current time

Definition at line 42 of file inc_cost.hpp.

6.4.4 Member Data Documentation

6.4.4.1 vec_type sac::inc_cost::m_mxdes

Definition at line 21 of file inc_cost.hpp.

6.4.4.2 state_type sac::inc_cost::m_x

Definition at line 20 of file inc_cost.hpp.

6.4.4.3 state_intp& sac::inc_cost::m_x_intp

Definition at line 19 of file inc_cost.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/inc_cost.hpp

6.5 sac::inc_state_cost Class Reference

`#include <traj_cost.hpp>`

Public Member Functions

- inc_state_cost (state_intp &rx_intp, mat_type &rmQ, Params &p)
- void operator() (const state_type &, state_type &dJdt, const double t)

6.5.1 Detailed Description

Computes the value of incremental trajectory tracking cost, $l(x(t))$ when called by a trajectory cost object.

Definition at line 10 of file traj_cost.hpp.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 sac::inc_state_cost::inc_state_cost (state_intp & *rx_intp,* mat_type & *rmQ,* Params & *p*) [inline]

Constructs a inc_state_cost object to computes the value of incremental trajectory tracking cost, $l(x(t))$ when called by a trajectory cost object.

Parameters

in	*rx_intp*	Reference to state interpolation object
in	*rmQ*	Weight matrix defining norm on incremental state tracking error
in	*p*	SAC parameters

Definition at line 27 of file traj_cost.hpp.

6.5.3 Member Function Documentation

6.5.3.1 void sac::inc_state_cost::operator() (const state_type & , state_type & *dJdt,* const double *t*) [inline]

Computes the value of incremental trajectory tracking cost, $l(x(t))$.

Parameters

in	*J*	The previous cost
out	*dJdt*	The current value of the incremental cost
in	*t*	The current time

Definition at line 44 of file traj_cost.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/traj_cost.hpp

6.6 sac::mode_insert_grad Class Reference

`#include <mode_insert_grad.hpp>`

Public Member Functions

- mode_insert_grad (state_intp &x_intp, state_intp &rho_intp, u2_optimal &u2Opt, Params &p)
- void operator() (const double t, double &dJdlam_curr)

6.6.1 Detailed Description

Stores the values of the mode insertion gradient, $\frac{dJ1}{d\lambda^+}(t) = \rho(t)^T[f(u_2^*(t)) - f(u_1(t))]$. Keeps references to user maintained state and co-state trajectory interpolation objects and $u_2^*(t)$ object so that the mode insertion graident automatically updates along with changes in trajectory and $u_2^*(t)$.

Definition at line 13 of file mode_insert_grad.hpp.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 sac::mode_insert_grad::mode_insert_grad (state_intp & *x_intp,* state_intp & *rho_intp,* u2_optimal & *u2Opt,* Params & *p*) [inline]

Todo Alex: make inputs const ref type

Initializes references to user maintained trajectory and control objects.

Parameters

in	*x_intp*	User maintained state interpolation object
in	*rho_intp*	User maintained co-state interpolation object
in	*u2Opt*	User maintained $u_2^*(t)$ object
in	*p*	SAC parameters

Definition at line 32 of file mode_insert_grad.hpp.

6.6.3 Member Function Documentation

6.6.3.1 void sac::mode_insert_grad::operator() (const double *t*, double & *dJdlam_curr*) [inline]

Computes the value of the mode insertion gradient, $\frac{dJ1}{d\lambda^+}(t)$.

$$\frac{dJ1}{d\lambda^+}(t) = \rho(t)^T[f(u_2^*(t)) - f(u_1(t))]$$

Parameters

in	*t*	The time at which to compute the mode insertion gradient
out	*dJdlam_curr*	The value of the mode insertion gradient

Definition at line 52 of file mode_insert_grad.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/mode_insert_grad.hpp

6.7 sac::Params Class Reference

`#include <params.hpp>`

Public Member Functions

- Params (const size_t xlen, const size_t ulen)
- size_t xlen () const
- size_t ulen () const
- double & T ()
- double & lam ()
- double & maxdt ()
- double & ts ()
- std::vector< std::vector< double > > & usat ()
- double & calc_tm ()
- bool & u2search ()
- mat_type & Q ()
- mat_type & P ()
- mat_type & R ()
- size_t & backtrack_its ()
- double & eps_cost ()
- vec_type & mxdes_tf ()

Public Attributes

- std::function< void(const double &, const state_type &, vec_type &)> x_des

  the desired trajectory at time t0 + T
- std::function< void(state_type &)> proj
- std::function< void(const state_type &, mat_type &)> gproj

6.7.1 Detailed Description

User interface for specifying parameters required by SAC

Definition at line 11 of file params.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 sac::Params::Params (const size_t *xlen*, const size_t *ulen*) [inline]

Parameters

in	*t*	the time at which to compute the incremental cost value
in	*x*	the state at which to compute the incremental cost value. Warning, this will be projected by the function.

Returns

incremental state cost

Computes the value of the derivative of the incremental cost, $D_x l(x)$.

Parameters

in	*t*	The current time
in	*mx*	The current state
out	*dldx*	The derivative $D_x l(x)$.

Constructor for Params class to hold SAC parameters.

Parameters

in	*xlen*	state vector dimension
in	*ulen*	control vector dimension

Definition at line 80 of file params.hpp.

6.7.3 Member Function Documentation

6.7.3.1 size_t& sac::Params::backtrack_its ( ) [inline]

Returns

A reference to the backtrack_its_ weight matrix for both getting and setting

Definition at line 212 of file params.hpp.

6.7.3.2 double& sac::Params::calc_tm ( ) [inline]

Returns

A reference to the time that each SAC control calculation is assumed to take. During this interval the previous control is applied while SAC computes the control to apply after this time has elapsed.

Definition at line 178 of file params.hpp.

6.7.3.3 double& sac::Params::eps_cost ( ) [inline]

Returns

A reference to the eps_cost_ for both getting and setting

Definition at line 217 of file params.hpp.

6.7.3.4 double& sac::Params::lam ( ) [inline]

Returns

A reference to the descent rate

Definition at line 152 of file params.hpp.

6.7.3.5 double& sac::Params::maxdt ( ) [inline]

Returns

A reference to the initial (max) duration for control application used in searching for a valid application duration

Definition at line 158 of file params.hpp.

6.7.3.6 vec_type& sac::Params::mxdes_tf ( ) [inline]

Returns

A reference to mxdes_tf_ for both getting and setting

Definition at line 222 of file params.hpp.

6.7.3.7 mat_type& sac::Params::P ( ) [inline]

Returns

A reference to the mP1_ weight matrix for both getting and setting

Definition at line 201 of file params.hpp.

6.7.3.8 mat_type& sac::Params::Q ( ) [inline]

get using: mat_type rQ = J_traj.Q(); get ref using: mat_type & rQ = J_traj.Q(); set using: param.Q() << 1000, 0, 0, 10;

Returns

A reference to the mQ_ weight matrix for both getting and setting

Definition at line 195 of file params.hpp.

6.7.3.9 mat_type& sac::Params::R ( ) [inline]

Returns

A reference to the mR_ weight matrix for both getting and setting

Definition at line 206 of file params.hpp.

6.7.3.10 double& sac::Params::T ( ) [inline]

get using: double T = param.T(); get ref using: double & T = param.T(); set using: param.T() = 0.5;

Returns

A reference to the prediction horizon

Definition at line 147 of file params.hpp.

6.7.3.11 double& sac::Params::ts ( ) [inline]

Returns

A reference to the sampling time. This specifies the time between feedback incorporation and the duration between consecutive control calculations

Definition at line 165 of file params.hpp.

6.7.3.12 bool& sac::Params::u2search ( ) [inline]

Returns

A reference to the boolean signifying whether SAC should search for an optimal time to apply each control action or if it should apply the control associated at the earlies feasible time (at the current time plus calc_tm seconds).

Definition at line 186 of file params.hpp.

6.7.3.13 size_t sac::Params::ulen ( ) const [inline]

Returns

An unsigned number representing the control dimension

Definition at line 139 of file params.hpp.

6.7.3.14 std::vector<std::vector<double> >& sac::Params::usat ( ) [inline]

Returns

A reference to a 2d vector holding the [max min] saturation values for each element of the control vector

Definition at line 171 of file params.hpp.

6.7.3.15 size_t sac::Params::xlen ( ) const [inline]

Returns

An unsigned number representing the state dimension

Definition at line 134 of file params.hpp.

6.7.4 Member Data Documentation

6.7.4.1 std::function<void(const state_type &, mat_type &)> sac::Params::gproj

Gradient of the state projection

Parameters

in	*x*	the state at which to take the gradient of the projection
out	*gproj_x*	the matrix storing the gradient of the projection

Definition at line 55 of file params.hpp.

6.7.4.2 std::function<void(state_type &)> sac::Params::proj

Projection to apply to the state matrix

Parameters

in,out	*x*	a state_type storing the current state

Definition at line 48 of file params.hpp.

6.7.4.3 std::function<void(const double &, const state_type &, vec_type &)> sac::Params::x_des

the desired trajectory at time t0 + T

Stores a user defined function that returns the desired trajectory

Parameters

in	*t*	time to evaluate desired trajectory
in	*x*	state at time x(t) for evaluation of desired trajectory
out	*mx_des*	an Eigen vec_type storing the desired trajectory

Definition at line 42 of file params.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/params.hpp

6.8 sac::push_back_state_and_time Struct Reference

`#include <master.hpp>`

Public Member Functions

- push_back_state_and_time (std::vector< state_type > &states, std::vector< double > &times)
- void operator() (const state_type &x, double t)

Public Attributes

- std::vector< state_type > & m_states
- std::vector< double > & m_times

6.8.1 Detailed Description

Observer class that stores states and times during integration in user specified containers of type std::vector< state_type >.

Definition at line 71 of file master.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 sac::push_back_state_and_time::push_back_state_and_time (std::vector< state_type > & *states*, std::vector< double > & *times*) [inline]

m_states(states) , m_times(times)

Initializes member variables to reference user specified containers to hold the states and times to be updated during integration.

Parameters

in,out	*states*	The container to hold the vector of states.
in,out	*times*	The container to hold the vector of times.

Definition at line 76 of file master.hpp.

6.8.3 Member Function Documentation

6.8.3.1 void sac::push_back_state_and_time::operator() (const state_type & *x*, double *t*) [inline]

Pushes states and times during integration.

Parameters

in	*x*	The state to push to the user specified container holding the vector of states.
in	*t*	The time to push to the user specified container holding the vector of times.

Definition at line 80 of file master.hpp.

6.8.4 Member Data Documentation

6.8.4.1 std::vector< state_type >& sac::push_back_state_and_time::m_states

Definition at line 73 of file master.hpp.

6.8.4.2 std::vector< double >& sac::push_back_state_and_time::m_times

Definition at line 74 of file master.hpp.

The documentation for this struct was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/master.hpp

6.9 sac::sac_step Class Reference

The class to carry out a sac control step.

`#include <sac_step.hpp>`

Public Member Functions

- sac_step (Params &p)
- void operator() (double &t0, state_type &x0, b_control &u1)

Public Attributes

- double J0
- double Jn
- double t_i
- double t_f
- double t_app
- double tf
- state_intp x_intp
- state_intp rho_intp
- state_type x0noU
- b_control u
- std::vector< state_type > x_vec
- std::vector< state_type > rho_vec
- std::vector< double > times
- std::vector< double > rho_times
- size_t its

Protected Member Functions

- virtual void SimInitXRho (const double &t0, const state_type &x0, const b_control &u1)
- virtual void SimNewX (const double &t0, const state_type &x0, const b_control &u1)

Protected Attributes

- double alpha_
- double min_val_
- double dtt_win_
- double dt_win_
- state_type x_
- state_type rho_
- state_type u_curr_
- sys_dynam xdot_
- adjoint rho_dot_
- vec_type m_mrho_tf_
- u2_optimal u2Opt_
- mode_insert_grad dJdlam_
- double dJdlam_curr_
- u2_cost cntrlCost_
- std::vector< double > lclMin_
- cost J1_
- iter_1d it1_1d_
- size_t j_
- size_t steps_
- Params & p_

6.9.1 Detailed Description

The class to carry out a sac control step.

Definition at line 7 of file sac_step.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 sac::sac_step::sac_step (Params & *p*) [inline]

Definition at line 35 of file sac_step.hpp.

6.9.3 Member Function Documentation

6.9.3.1 void sac::sac_step::operator() (double & *t0*, state_type & *x0*, b_control & *u1*) [inline]

Performs an iteration of SAC control.

Parameters

in,out	*t0*	initial time associated with state vector input x0. This get updated by one sample time to t0=t0+ts after stepper completes.
in,out	*x0*	initial state vector to be integrated forward after one iteration of SAC. The stepper integrates x0 from time t0 to time t0+ts based on SAC controls.
in,out	*u1*	default control value to apply from t0 to t0+calc_tm. The stepper computes the new SAC control to appy from t0+calc_tm to t0+calc_tm+ts and returns it in u1.

Definition at line 60 of file sac_step.hpp.

6.9.3.2 void sac::sac_step::SimInitXRho (const double & *t0*, const state_type & *x0*, const b_control & *u1*) [inline], [protected], [virtual]

Definition at line 134 of file sac_step.hpp.

6.9.3.3 void sac::sac_step::SimNewX (const double & *t0*, const state_type & *x0*, const b_control & *u1*) [inline], [protected], [virtual]

Definition at line 149 of file sac_step.hpp.

6.9.4 Member Data Documentation

6.9.4.1 double sac::sac_step::alpha_ [protected]

Definition at line 10 of file sac_step.hpp.

6.9.4.2 u2_cost sac::sac_step::cntrlCost_ [protected]

Definition at line 15 of file sac_step.hpp.

6.9.4.3 mode_insert_grad sac::sac_step::dJdlam_ [protected]

Definition at line 14 of file sac_step.hpp.

6.9.4.4 double sac::sac_step::dJdlam_curr_ [protected]

Definition at line 15 of file sac_step.hpp.

6.9.4.5 double sac::sac_step::dt_win_ [protected]

Definition at line 10 of file sac_step.hpp.

6.9.4.6 double sac::sac_step::dtt_win_ [protected]

Definition at line 10 of file sac_step.hpp.

6.9.4.7 iter_1d sac::sac_step::it1_1d_ [protected]

Definition at line 17 of file sac_step.hpp.

6.9.4.8 size_t sac::sac_step::its

Definition at line 33 of file sac_step.hpp.

6.9.4.9 double sac::sac_step::J0

Definition at line 28 of file sac_step.hpp.

6.9.4.10 cost sac::sac_step::J1_ [protected]

Definition at line 17 of file sac_step.hpp.

6.9.4.11 size_t sac::sac_step::j_ [protected]

Definition at line 17 of file sac_step.hpp.

6.9.4.12 double sac::sac_step::Jn

Definition at line 28 of file sac_step.hpp.

6.9.4.13 std::vector<double> sac::sac_step::lclMin_ [protected]

Definition at line 16 of file sac_step.hpp.

6.9.4.14 vec_type sac::sac_step::m_mrho_tf_ [protected]

Definition at line 13 of file sac_step.hpp.

6.9.4.15 double sac::sac_step::min_val_ [protected]

Definition at line 10 of file sac_step.hpp.

6.9.4.16 Params& sac::sac_step::p_ [protected]

Definition at line 18 of file sac_step.hpp.

6.9.4.17 state_type sac::sac_step::rho_ [protected]

Definition at line 11 of file sac_step.hpp.

6.9.4.18 adjoint sac::sac_step::rho_dot_ [protected]

Definition at line 12 of file sac_step.hpp.

6.9.4.19 state_intp sac::sac_step::rho_intp

Definition at line 29 of file sac_step.hpp.

6.9.4.20 std::vector<double> sac::sac_step::rho_times

Definition at line 32 of file sac_step.hpp.

6.9.4.21 std::vector<state_type> sac::sac_step::rho_vec

Definition at line 31 of file sac_step.hpp.

6.9.4.22 size_t sac::sac_step::steps_ [protected]

Definition at line 17 of file sac_step.hpp.

6.9.4.23 double sac::sac_step::t_app

Definition at line 28 of file sac_step.hpp.

6.9.4.24 double sac::sac_step::t_f

Definition at line 28 of file sac_step.hpp.

6.9.4.25 double sac::sac_step::t_i

Definition at line 28 of file sac_step.hpp.

6.9.4.26 double sac::sac_step::tf

Definition at line 28 of file sac_step.hpp.

6.9.4.27 std::vector<double> sac::sac_step::times

Definition at line 32 of file sac_step.hpp.

6.9.4.28 b_control sac::sac_step::u

Definition at line 30 of file sac_step.hpp.

6.9.4.29 u2_optimal sac::sac_step::u2Opt_ [protected]

Definition at line 14 of file sac_step.hpp.

6.9.4.30 state_type sac::sac_step::u_curr_ [protected]

Definition at line 11 of file sac_step.hpp.

6.9.4.31 state_type sac::sac_step::x0noU

Definition at line 30 of file sac_step.hpp.

6.9.4.32 state_type sac::sac_step::x_ [protected]

Definition at line 11 of file sac_step.hpp.

6.9.4.33 state_intp sac::sac_step::x_intp

Definition at line 29 of file sac_step.hpp.

6.9.4.34 std::vector<state_type> sac::sac_step::x_vec

Definition at line 31 of file sac_step.hpp.

6.9.4.35 sys_dynam sac::sac_step::xdot_ [protected]

Definition at line 12 of file sac_step.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/sac_step.hpp

6.10 sac::state_intp Class Reference

`#include <state_intp.hpp>`

Public Member Functions

- state_intp (std::vector< state_type > &states, std::vector< double > &times)
- state_intp (std::vector< state_type > &states, std::vector< double > &times, size_t xlength)
- void operator() (const double t_intp, state_type &x_intp)
- void update (std::vector< state_type > &states, std::vector< double > &times)
- double begin ()
- double end ()

Public Attributes

- std::vector< state_type > * m_states
- std::vector< double > * m_times
- size_t m_xlen

6.10.1 Detailed Description

Class holds the address of state_types and times vectors. It uses interpolation to provide the state at a specified time.

Definition at line 10 of file state_intp.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 sac::state_intp::state_intp (std::vector< state_type > & *states*, std::vector< double > & *times*) [inline]

Todo Alex: make inputs const ref type

Initializes pointers to user maintained state_type and times vectors.

Parameters

in	*states*	Vector of states sampled at different points in time
in	*times*	Vector of times corresponding to the sampled states

Definition at line 26 of file state_intp.hpp.

6.10.2.2 sac::state_intp::state_intp (std::vector< state_type > & *states*, std::vector< double > & *times*, size_t *xlength*) [inline]

Todo Alex: make inputs const ref type

Initializes pointers to user maintained state_type and times vectors.

Parameters

in	*states*	Vector of states sampled at different points in time
in	*times*	Vector of times corresponding to the sampled states
in	*xlength*	The length of state x(t)

Definition at line 47 of file state_intp.hpp.

6.10.3 Member Function Documentation

6.10.3.1 double sac::state_intp::begin ( ) [inline]

Returns

The first element of the vector of times.

Definition at line 92 of file state_intp.hpp.

6.10.3.2 double sac::state_intp::end ( ) [inline]

Returns

The last element of the vector of times.

Definition at line 97 of file state_intp.hpp.

6.10.3.3 void sac::state_intp::operator() (const double *t_intp*, state_type & *x_intp*) [inline]

Applies linear interpolation to provide the state at the specified time.

Parameters

in	*t_intp*	The time to interpolate the state
out	*x_intp*	The state at the specified time

Definition at line 60 of file state_intp.hpp.

6.10.3.4 void sac::state_intp::update (std::vector< state_type > & *states*, std::vector< double > & *times*) [inline]

Todo Alex: make inputs const ref type

Updates the state_type and times vectors pointed to.

Parameters

in	*states*	Vector of states sampled at different points in time
out	*times*	Vector of times corresponding to the sampled states

Definition at line 83 of file state_intp.hpp.

6.10.4 Member Data Documentation

6.10.4.1 std::vector< state_type >* sac::state_intp::m_states

Definition at line 16 of file state_intp.hpp.

6.10.4.2 std::vector< double >* sac::state_intp::m_times

Definition at line 17 of file state_intp.hpp.

6.10.4.3 size_t sac::state_intp::m_xlen

Definition at line 18 of file state_intp.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/state_intp.hpp

6.11 sac::traj_cost Class Reference

`#include <traj_cost.hpp>`

Public Member Functions

- traj_cost (state_intp &rx_intp, std::vector< state_type > &ru2list, std::vector< state_type > &rTiTappTf, const vec_type &rmxdes_tf, Params &p)
- traj_cost (state_intp &rx_intp, std::vector< state_type > &ru2list, std::vector< state_type > &rTiTappTf, Params &p)
- size_t compute_cost (const double t0, const double tf)
- void print ()
- double get_cost ()
- void get_costs (std::vector< double > &cost_vec)
- mat_type & Q ()
- mat_type & P ()
- mat_type & R ()

6.11.1 Detailed Description

Evaluates the trajectory cost over a defined horizon from state and control matrices. $J_1 = \frac{1}{2}\int_{t_0}^{t_f}(x-x_{des})^T Q\,(x-x_{des}) + u^T R\,u\,dt + \frac{1}{2}(x-x_{des})^T P\,(x-x_{des})$ Keeps references to state interpolator so that changes in state trajectory are automatically accounted for.

Definition at line 70 of file traj_cost.hpp.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 sac::traj_cost::traj_cost (state_intp & *rx_intp*, std::vector< state_type > & *ru2list*, std::vector< state_type > & *rTiTappTf*, const vec_type & *rmxdes_tf*, Params & *p*) [inline]

Todo Alex: Make constructor reference and pointer inputs const.

Constructs a traj_cost object from a state interpolation object, desired state trajectory, and the list of controls and application times.

Parameters

in	*rx_intp*	Reference to state interpolation object
in	*ru2list*	Reference to list of applied control
in	*rTiTappTf*	Reference to list of application times for u2
in	*rmxdes_tf*	Reference to $x_{des}(t_f)$
in	*p*	SAC parameters

Definition at line 96 of file traj_cost.hpp.

6.11.2.2 sac::traj_cost::traj_cost (state_intp & *rx_intp*, std::vector< state_type > & *ru2list*, std::vector< state_type > & *rTiTappTf*, Params & *p*) [inline]

Todo Alex: Make constructor reference and pointer inputs const.

Constructs a traj_cost object from a state interpolation object, desired state trajectory, and the list of controls and application times. This alternate constructor assumes no terminal cost on trajectory.

Parameters

in	*rx_intp*	Reference to state interpolation object
in	*ru2list*	Reference to list of applied control
in	*rTi↩ TappTf*	Reference to list of application times for u2
in	*p*	SAC parameters

Definition at line 128 of file traj_cost.hpp.

6.11.3 Member Function Documentation

6.11.3.1 size_t sac::traj_cost::compute_cost (const double *t0*, const double *tf*)

The function computes the integral of $l(x(t), u(t))$ and appends it to the terminal cost to return cost $J = \int_{t_0}^{t_f} l(x(t), u(t))\, dt + m(x(t_f))$.

Parameters

in	*t0*	The initial time for integration.
in	*tf*	The final time for integration.

Returns

The number of integration steps required.

Definition at line 200 of file traj_cost.hpp.

6.11.3.2 double sac::traj_cost::get_cost ( ) [inline]

Returns

The trajectory cost computed from the last call to compute_cost()

Definition at line 159 of file traj_cost.hpp.

6.11.3.3 void sac::traj_cost::get_costs (std::vector< double > & *cost_vec*) [inline]

Parameters

out	*cost_vec*	Vector of 1) total trajectory cost 2) state cost 3) control cost and 4) terminal cost

Definition at line 165 of file traj_cost.hpp.

6.11.3.4 mat_type& sac::traj_cost::P ( ) [inline]

Returns

A reference to the mP1_ weight matrix for both getting and setting

Definition at line 184 of file traj_cost.hpp.

6.11.3.5 void sac::traj_cost::print ( ) [inline]

Prints the integrated control and state tracking costs, the terminal cost, and the total cost of the trajectory as of the last call to compute_cost( ). Results are outputed to std out.

Definition at line 148 of file traj_cost.hpp.

6.11.3.6 mat_type& sac::traj_cost::Q ( ) [inline]

get using: mat_type rQ = J_traj.Q(); get ref using: mat_type & rQ = J_traj.Q(); set using: J_traj.Q() << 1000, 0, 0, 10;

Returns

A reference to the mQ_ weight matrix for both getting and setting

Definition at line 179 of file traj_cost.hpp.

6.11.3.7 mat_type& sac::traj_cost::R ( ) [inline]

Returns

A reference to the mR_ weight matrix for both getting and setting

Definition at line 189 of file traj_cost.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/traj_cost.hpp

6.12 sac::u2_cost Class Reference

`#include <u2_cost.hpp>`

Public Member Functions

- u2_cost (u2_optimal &u2Opt, mode_insert_grad &dJdlam, Params &p)
- double operator() (const double t)

6.12.1 Detailed Description

Provides a cost function of time that can be optimized to find the best time to apply the optimal control law u2$*$. e.g. $cost(t) = \sqrt{u_2^*(t)^T u_2^*(t)} + \frac{dJ_1}{d\lambda^+}(t) + t^{1.6}$

Definition at line 11 of file u2_cost.hpp.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 sac::u2_cost::u2_cost (u2_optimal & *u2Opt,* mode_insert_grad & *dJdlam,* Params & *p*) [inline]

Todo Alex: see about converting inputs to const ref types.

Initializes references to user defined objects $u_2^*(t)$ and $\frac{dJ_1}{d\lambda^+}(t)$ which automatically update along with updates in trajectory.

Parameters

in	*u2Opt*	The object storing the values of control $u_2^*(t)$
in	*dJdlam*	The object storing the values of mode insertion gradient $\frac{dJ_1}{d\lambda^+}(t)$
in	*p*	SAC parameters

Definition at line 31 of file u2_cost.hpp.

6.12.3 Member Function Documentation

6.12.3.1 double sac::u2_cost::operator() (const double *t*) [inline]

Returns the value of a cost function at the specified time. The cost function can be searched to find the best time to apply $u_2^*(t)$.

$$cost(t) = \sqrt{u_2^*(t)^T u_2^*(t)} + \frac{dJ_1}{d\lambda^+}(t) + t^{1.6}$$

Parameters

in	*t*	The time at which to evaluate the cost

Returns

The value of the cost at the specified time

Definition at line 45 of file u2_cost.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/u2_cost.hpp

6.13 sac::u2_optimal Class Reference

`#include <u2_optimal.hpp>`

Public Member Functions

- u2_optimal (state_intp &x_intp, state_intp &rho_intp, double &alpha, Params &p)
- void operator() (const double t, state_type &u2Opt_curr)

6.13.1 Detailed Description

Stores the optimal switching control, $u_2^*(t)$. Keeps references to user maintained state and co-state trajectory interpolation objects so that $u_2^*(t)$ automatically updates along with changes in trajectory.

$$u_2^*(t) = (\Lambda + R)^{-1} [\Lambda\, u_1(t) + h(x(t))^T \rho(t)\, \alpha_d]$$

where $\Lambda = h(x(t))^T \rho(t) \rho(t)^T h(x(t))$ and $h(x(t)) = \frac{\partial f_1}{\partial u_1}$. The dynamics, f_1, should be in control affine form.

Definition at line 16 of file u2_optimal.hpp.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 sac::u2_optimal::u2_optimal (state_intp & *x_intp*, state_intp & *rho_intp*, double & *alpha*, Params & *p*) [inline]

Todo Alex: make inputs const ref type

Initializes references to user maintained trajectory objects and the desired rate of change of a trajectory tracking cost functional.

Parameters

in	*x_intp*	User maintained state interpolation object
in	*rho_intp*	User maintained co-state interpolation object
in	*alpha*	User specified desired change in cost functional relative to the duration of activiation of $u_2^*(t)$. i.e. $\frac{\Delta J_1}{\Delta t}$.
in	*p*	SAC parameters

Definition at line 42 of file u2_optimal.hpp.

6.13.3 Member Function Documentation

6.13.3.1 void sac::u2_optimal::operator() (const double *t*, state_type & *u2Opt_curr*) [inline]

Todo Alex: incorporate u1

Computes the value of the optimal switching control, $u_2^*(t)$.

$$u_2^*(t) = \ (\Lambda + R)^{-1} \left[\Lambda \, u_1(t) + h(x(t))^T \rho(t) \, \alpha_d\right]$$

Parameters

in	*t*	The time at which to compute the mode insertion gradient
out	*u2Opt_curr*	The value of the optimal switching control

Definition at line 67 of file u2_optimal.hpp.

The documentation for this class was generated from the following file:

- /home/alex/Documents/code/C/SAC_v2/lib/src/u2_optimal.hpp

Chapter 7

File Documentation

7.1 /home/alex/Documents/code/C/SAC_v2/lib/src/adjoint.hpp File Reference

Classes

- class sac::adjoint

Namespaces

- sac

7.2 /home/alex/Documents/code/C/SAC_v2/lib/src/b_control.hpp File Reference

Classes

- class sac::b_control

Namespaces

- sac

7.3 /home/alex/Documents/code/C/SAC_v2/lib/src/cost.hpp File Reference

Classes

- class sac::cost

Namespaces

- sac

7.4 /home/alex/Documents/code/C/SAC_v2/lib/src/inc_cost.hpp File Reference

Classes

- class sac::inc_cost

Namespaces

- sac

7.5 /home/alex/Documents/code/C/SAC_v2/lib/src/master.hpp File Reference

```
#include <algorithm>
#include <vector>
#include <boost/numeric/odeint.hpp>
#include <fstream>
#include <Eigen/Core>
#include <Eigen/Dense>
#include <params.hpp>
#include <b_control.hpp>
#include <state_intp.hpp>
#include <sys_dynam.hpp>
#include <sys_lin.hpp>
#include <u2_optimal.hpp>
#include <mode_insert_grad.hpp>
#include <u2_cost.hpp>
#include <inc_cost.hpp>
#include <cost.hpp>
#include <adjoint.hpp>
#include <sac_step.hpp>
```

Classes

- struct sac::push_back_state_and_time

Namespaces

- sac

Macros

- #define PI (3.14159)

Typedefs

- typedef std::vector< double > sac::state_type
- typedef std::vector< double >::iterator sac::iter_1d
- typedef std::vector< state_type >::iterator sac::iter_2d
- typedef Eigen::MatrixXd sac::mat_type
- typedef Eigen::MatrixXd sac::vec_type

Functions

- template<class T >
  void sac::State2Mat (const state_type &s, T &matOut)
- template<class T >
  void sac::Mat2State (const T &mat, state_type &sOut)
- template<class Scalar >
  void sac::AngleWrap (Scalar &theta)
- template<class T , class Scalar >
  void sac::MinSearch (T &fcost, Scalar t0, Scalar tf, std::vector< Scalar > &lclMin, Scalar dt, Scalar eps)
- template<class T , class InputIterator , class Function >
  InputIterator sac::get_min (InputIterator first, InputIterator last, Function &fn, T &min)
- size_t sac::simX (sys_dynam &xdot, state_type &x0, double t0, double tf, std::vector< state_type > &x_vec, std::vector< double > &times)
- size_t sac::simRho (adjoint &rho_dot, state_type &rho_Tf, double t0, double tf, std::vector< state_type > &rho_vec, std::vector< double > &rho_times)
- template<typename T >
  int sac::sgn (T val)
- template<class T , class Scalar >
  Scalar sac::GoldenSection (T &fcost, Scalar a, Scalar b, Scalar c, Scalar &dt, Scalar &eps)
- template<class T >
  void sac::SaveVec (const T &src, const char *outputFilename)

7.5.1 Detailed Description

The master include file for the library.

7.5.2 Macro Definition Documentation

7.5.2.1 #define PI (3.14159)

Definition at line 22 of file master.hpp.

7.6 /home/alex/Documents/code/C/SAC_v2/lib/src/mode_insert_grad.hpp File Reference

Classes

- class sac::mode_insert_grad

Namespaces

- sac

7.7 /home/alex/Documents/code/C/SAC_v2/lib/src/params.hpp File Reference

```
#include <functional>
```

Classes

- class sac::Params

Namespaces

- sac

7.8 /home/alex/Documents/code/C/SAC_v2/lib/src/sac_step.hpp File Reference

Classes

- class sac::sac_step

  The class to carry out a sac control step.

Namespaces

- sac

7.9 /home/alex/Documents/code/C/SAC_v2/lib/src/state_intp.hpp File Reference

Classes

- class sac::state_intp

Namespaces

- sac

7.10 /home/alex/Documents/code/C/SAC_v2/lib/src/traj_cost.hpp File Reference

Classes

- class sac::inc_state_cost
- class sac::traj_cost

Namespaces

- sac

7.11 /home/alex/Documents/code/C/SAC_v2/lib/src/u2_cost.hpp File Reference

Classes

- class sac::u2_cost

Namespaces

- sac

7.12 /home/alex/Documents/code/C/SAC_v2/lib/src/u2_optimal.hpp File Reference

Classes

- class sac::u2_optimal

Namespaces

- sac

Index