TEB Package

0.1

Generated by Doxygen 1.8.6

Thu Mar 5 2015 14:26:41

Contents

Chapter 1

Timed-Elastic-Band Package Documentation

1.1 Overview

This project contains the implementation of a predictive controller and trajectory planner based on Timed-Elastic-Bands (TEB) (see [5]).

The TEB merges the states, control inputs and time intervals into a joint trajectory representation which enables planning of time-optimal trajectories in the context of model predictive control. Model predictive control integrates the planning of the optimal trajectory with state feedback in the control loop. The TEB approach formulates the fixed horizon optimal control problem for point-to-point transitions as a nonlinear program.

1.2 TEB Optimization Problem

1.2.1 Problem Formulation

The TEB joint trajectory representation is defined as follows:

$$\mathscr{B} := \{\mathbf{x}_1, \mathbf{u}_1, \mathbf{x}_2, \mathbf{u}_2, \ldots, \mathbf{x}_{n-1}, \mathbf{u}_{n-1}, \mathbf{x}_n, \Delta T\}$$

$\mathbf{x}_i \in \mathbb{R}^p$ and $\mathbf{u}_i \in \mathbb{R}^q$ are the discretized state and control input vectors respectively. ΔT denotes the time difference required to transit from the *i-th* state to the consecutive one. Currently, this time difference is obtained from applying finite differences to the system dynamics equation (by replacing continuous derivatives).

This project considers nonlinear programs of the following form to optimize the TEB sequence $\mathscr{B}$.

$$V^*(\mathscr{B}) = \min f(\mathscr{B}) \tag{1.1}$$

$$\text{s.t.:}$$

$$\mathbf{x}_1 = \mathbf{x}_s \tag{1.2}$$

$$\mathbf{x}_n \approx \mathbf{x}_f \tag{1.3}$$

$$\mathbf{h}_k(\mathscr{B}) = \mathbf{0} \quad k \in \mathscr{E} \tag{1.4}$$

$$\mathbf{g}_k(\mathscr{B}) \leq \mathbf{0} \quad k \in \mathscr{I} \tag{1.5}$$

in which $f(\mathscr{B})$ denotes the objective/cost function. Equation (2) ensures that the nonlinear program constitutes at least an initial value problem. The final constrained equation (3) is not mendatory. But if it is not provided, the final state behavior should be specified using dedicated final state costs as part of $f(\mathscr{B})$. $\mathscr{E}$ and $\mathscr{I}$ denote the indices sets of equality and inequality constraints respectively.

For many problems the structure of the underlying optimization problem is sparse. To exploit these local relationships between states and control inputs, the optimization problem is formulated as a hyper-graph according to the formulism in [2]. The user has to define edges that represent "local" cost functions according to teb::BaseEdge and its subclasses. They can be added to the TEB controller by overriding teb::TebController::customOptimizationGraph(). A lot of common optimization and model predictive control problems are already implemented as Edge (especially for nonlinear dynamic systems in state-space representation. Refer to teb::SystemDynamics and the example programs for further information.

1.2.2 Solution of the TEB Optimization Problem

Solving the nonlinear program (1)-(5) requieres the utilization of dedicated solvers that support constraints. The solver classes are constantly expanded as subclasses of teb::BaseSolver.

An approximation to the solution can by computed efficiently using "Least Squares Solvers" and "Soft Constraints". Least Square Optimziation Problem require squared objectives. The solver class teb::BaseSolverLeastSquares transforms equation (1) to $V^*(\mathscr{B}) = \min f(\mathscr{B})^2$. Hard constraints (4) and (5) are transformed to soft constraints. Refer to the specific class description. teb::SolverLevenbergMarquardtEigenDense optimizes this transformed "least-squares" problem using the Levenberg-Marquardt algorithm (Dense means, that no sparse structure is exploited for the solution of the linear system. But only for the calculation of Jacobians due to the hyper-graph representation).

1.3 Further Information

Fore more detailed information about the Timed-Elastic-Band approach and applications refer to [3] [4] [1] [2].

1.3.1 Download

Refer to the download section.

1.3.2 Installation

Refer to the installation instructions.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Remarks

For research only.

Chapter 2

Download TEB-Package

For Users

The project source code is managed by a subversion control system (SVN). Find the svn repository at `https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/`.

To get the newest (experimental and maybe unstable) version checkout the *trunk* folder using:

```
svn co https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/trunk teb_package
```

Find versions tagged as stable or *working and tested with windows* in the *tags* folder.

They can be checked out in a similar way (**Warning**: Never commit changes to an existing tagged version, copy it to your own branch instead).

To check out a specific branch of the project, consider the *branches* folder.

For Developers

Work on the trunk

Checkout the trunk:

```
svn co https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/trunk teb_package
```

Modify or add code and check if you can compile the code successfully. Check the status of your changes:

```
svn st
```

Check files marked with a question mark, whether they should be added to the repository (use `svn add <path/filename>`).

Never add build files and temporary files to the repository!

Afterwards commit your changes to the trunk:

```
svn ci -m "This should be a comprehensive commit message for the log"
```

Next time you continue working on the trunk, don't forget to call `svn up` to get the latest updates.

In case of conflicts, please resolve them by merging the code or contact the author of the previous (conflicting) commit!

Create your own branch

Way 1:

Create the branch directly on the server:

```
svn cp https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/trunk \
https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/branches/mybranch
```

Now checkout your newly created branch. If you have no working copy of the project yet, use:

```
svn co https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/branches/mybranch teb_mybranch
```

If you already have a working copy (e.g. the trunk folder) call

```
svn switch https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/branches/mybranch
```

in the root of your trunk folder.

Way 2:

In case you checked out the complete project (including all branches and tags) using:

```
svn co https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/ teb_package
```

Now create a new branch while copying all project files (internally svn is smart enough to prevent double disk space usage):

```
mkdir branches/mybranch
svn add branches/mybranch
svn cp trunk branches/mybranch  # Or copy a tag from the tags folder instead of the trunk to switch to the bra
```

Synchronizing your branch with the trunk

To merge the most recent updates from the trunk into your own branch do the following.

First check what files have been changed compared to your branch:

```
svn st -u
```

Merge the diferences in your local working directory:

```
svn merge https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/trunk
```

Often it is recommended to add `--dry-run` to the argument list in order to check for possible conflicts. See the remarks below on manually resolving conflicts.

Finally commit your merged code to your branch.

Merging a branch into the trunk

First synchronize your branch with the trunk and commit changes like mentioned before.

Now checkout the trunk to a dedicated directory.

Make sure that you do not have any local changes on the trunk (`svn up; svn st`).

Merge the (already synchronized) branch "mybranch" into the trunk using:

```
svn merge --reintegrate https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/branches/mybranch
```

Finally, commit your merged trunk to the server.

Click here and here for more information about merging.

Information about your working directory

Get information on your current revision and if you are working on the trunk or a branch using

```
svn info
```

Reverting local changes

Undo changes that are not yet committed:

```
svn revert <file>
```

Or undo everything (switch to the main directory first):

```
svn revert --recursive .
```

Correcting commit log messages

If you want to correct a previously made commit message, call:

```
svn propedit svn:log --revprop -r <REV-NR> https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/
```

where <REV-NR> denotes the revision number (you can get revision numbers via `svn log`).

Resolving conflicts

If svn was not able to solve conflicts automatically, try to open the files marked as conflicted and searh for lines starting with "<<<<<<<<" and ">>>>>>>>". Edit them manually and afterwards call

```
svn resolved <file>
```

Useful symbolic revision names

For some revisions, special symbolic names can be utilized:

- HEAD: latest (youngest) revision on the server
- BASE: the revision your checkout was last updated against
- COMMITTED: last revision a file or directory was changed
- PREV: the last revision the file or directory was changed immediatley before COMMITTED

Chapter 3

Install TEB-Package

Ubuntu

Prerequisites:

- *CMake:* Required to create the platform depenent build files (`sudo apt-get install cmake`).
- *Gnuplot* (optional, but recommanded): Required for plotting and visualization (`sudo apt-get install gnuplot gnuplot-x11`).
- *Doxygen* (optional): Necessary, if the documentation should be compiled (`sudo apt-get install doxygen`).
- *Tex Live* (optional): Only necessary to create a pdf documentation from the latex source generated by doxygen.
- *Matlab* (optional): Necessary only if the user wants to create mex-files and s-functions for the usage in Matlab and Simulink.
- *kDevelop* (optional): IDE for developers. It supports the import CMakeLists.txt directly (`sudo apt-get install kdevelop`).
- *SuiteSparse* (optional): Enables the usage of a wider range of sparse direct linear solvers (`sudo apt-get install libsuitesparse-dev`).

Usage:

TODO

Doxygen:

After the project is generated using CMake, cd into the ./build folder and call `make doc` from the terminal. Now a HTML and a LATEX folder should be created inside the doc folder. To create a PDF from the LATEX source directly, cd to ./doc/LATEX and type in your terminal: `make pdf` (Tex Live including pdflatex required!).

Mac OSX

Prerequisites:

- *XCode:* Download the newest x-code version from the app store or from the OS X developer page.
- *CMake:* Required to create the platform depenent build files (`http://www.cmake.org`).
- *Gnuplot* (optional, but recommanded): Required for plotting and visualization (`http://gnuplot.info`)
- *Doxygen* (optional): Necessary, if the documentation should be compiled (`http://www.stack.nl/~dimitri/doxygen/download.html`).

- *MacTex* (optional): Only necessary to create a pdf documentation from the latex source generated by doxygen.
- *Matlab* (optional): Necessary only if the user wants to create mex-files and s-functions for the usage in Matlab and Simulink.

Remarks: For me (Christoph), x11 (xquartz) was not working if gnuplot was called from default terminal. It worked only in xterm. Since xcode starts the default terminal, I decided to use aquaterm in the TEBPlotter class:

```
install xcode with command line tools
install homebrew
install aquaterm
terminal: brew install gnuplot --with-aquaterm
test installation:
terminal: gnuplot
plot x
```

Usage:

Open CMake and select the CMakeLists.txt inside the project folder. Select x-code as the underlying compiler, configure and generate the project into a "build" folder. You can find the x-code project files now inside the newly created build folder.

Doxygen:

After the project is generated using CMake, cd into the ./doc folder and call `doxygen doxygen.config` from the terminal. Now a HTML and a LATEX folder should be created inside the doc folder. To create a PDF from the LATEX source directly, cd to ./doc/LATEX and type in your terminal: `make pdf` (MacTex including pdflatex required!).

Windows

Prerequisites:

- *Visual Studio 2013+:* Download Visual Studio (At least Version 12 / 2013)
- *CMake:* Required to create the platform depenent build files (`http://www.cmake.org`).
- *Gnuplot* (optional, but recommanded): Required for plotting and visualization (`http://gnuplot.info`)
- *Doxygen* (optional): Necessary, if the documentation should be compiled (`http://www.stack.nl/~dimitri/doxygen/download.html`).
- *MikTex* (optional): Only necessary to create a pdf documentation from the latex source generated by doxygen.
- *Matlab* (optional): Necessary only if the user wants to create mex-files and s-functions for the usage in Matlab and Simulink.

Remarks: Gnuplot has to be included in Windows Path variable:

- Windows 8: System Control -> Extended System Settings -> Advanced -> Environment variables. Modify the "System Variable" PATH and append the Gnuplot binary directory. E.g.: PATH;C:\Program Files\gnuplot\bin\

Usage:

Open CMake and select the C++ project folder. Select Visual Studio 12 (2013) 64 Bit as the underlying compiler, configure and generate the project into a specified "build" folder. **Caution:** We observed problems using 32 Bit, therefore we recommend to select 64 Bit Visual Studio!! You can find the Visual-Studio project files now inside the newly created build folder.

Chapter 4

Solver Overview

The following list contains descriptions and small guidances about the solver backends implemented or interfaced within this package.

Least-Squares Solver

Least-Square solvers minimize only quadratic functions without constraints $V^*(\mathscr{B}) = \min f^2(\mathscr{B})$.

Advantages of the least-squares solver are the amazing performance, the robustness (Levenberg-Marquardt) and even after a few iterations, a suitable solution is found often. A drawback is the lack of hard constraints. This implementation transforms equality and inequality constraints automatically into soft constraints (refer to the class description).

Due to the prequisite that only squared function can be minimized, we need to keep the following important definitions/conventions in mind:

- The objective functions are always defined without squaring the values: $f(\mathscr{B})$ than $f^2(\mathscr{B})$.
- The solver will square the values internally (this definition is much more efficient than specifying the squared values).
- Constraints are squared internally as well using the soft-constraint approximation.
- To make the MPC optimization problem simultaneously compatible to Non-Linear Program Solvers (see below), the designer of an objective function (given as an edge type for the hyper-graph, see the mobile robot example) can specify some information about the function type (see Enumeration teb::FUNCT_TYPE). This can be helpful for the hessian calculation as well, since a linear function has a zero hessian. The function types are defined as follow:
  – *Nonlinear*: The function is nonlinear (it will be squared by LeastSquares solver, but not by other solvers)
  – *Nonlinear_Squared*: The function is nonlinear (it will be squared by all solvers)
  – *Nonlinear_Once_Diff*: The hessian of this function is always zero (and it will be squared only by Least-Square solvers)
  – *Quadratic*: The function will be squared only by LeastSquares solvers (-> Power of 4 !!!)
  – *Linear*: The function will be squared only by LeastSquares solvers (zero hessian)
  – *LinearSquared*: The function will be squared by all solvers (zero hessian)

Levenberg-Marquardt Dense (teb::SolverLevenbergMarquardtEigenDense)

This solver constitutes a robust least-squares optimization algorithm. The Levenberg-Marquardt algorithm is similar to the one used in [2]. The underlying linear system is solved using the Eigen framework.

Levenberg-Marquardt Sparse (teb::SolverLevenbergMarquardtEigenSparse)

This solver implements the sparse version of the Levenberg-Marquardt algorithm mentioned above. Jacobians and Hessians are internally treaten as sparse matrices, that are effiencly constructed using the hyper-graph formulation of the underlying optimization problem. The resulting sparse linear system is solved using the Eigen framework (sparse modules). If CMake is able to find SuiteSparse/Cholmod libraries (see Install TEB-Package), the cholmod solver is utilized for solving the sparse linear system as long as `!defined(FORCE_EIGEN_SOLVER)`.

Nonlinear Program Solver

Sequential-Quadratic-Programming (SQP) Dense with qpOASES (teb::SolverSQPDense)

The solver implementation is still in progress, documantation will be added in the future.

NLOPT Solver Interface (teb::SolverNloptPackage)

The solver implementation is still in progress, documantation will be added in the future.

Chapter 5

Matlab Interface

The following instructions give you an introduction on how to interface this package with Matlab and Matlab Simulink. The procedure is still experimental, please inform us about bugs and/or possible solutions.

Way 1: Compile MEX and S-Function wrapper using CMAKE

Advantages:

- Simple solution
- Allows the library *teb_package.lib* to be compiled either in Debug or Release mode

Disadvantages:

- In order to use the wrappers within *Matlab's/Simulink's RealTime Workshop*, the code must be compilable with Matlab. In this case refer to *Way 2* instead.

You can directly include MEX or S-Function wrapper functions (defined in separate ∗.cpp files) into the CMAKE build progress. See the *CMakeLists.txt* for examples.

If the projects are stored in the example folder, add them to:

```
SET(MEX_EXAMPLE_NAMES # matlab mex example files
   matlab_interface/integrator_system_mex
   # Add files without .cpp extension here, attach folder name as a prefix
)

SET(SFUN_EXAMPLE_NAMES # matlab simulink mex example files
   matlab_interface/integrator_system_sfun
   # Add files without .cpp extension here, attach folder name as a prefix
)
```

If you use a separate project, add them using ADD_MEX_FILE macro (see *cmake/macros* directory) and link against the *teb_library.lib*. You need to include Matlabs source and libs (see *FindMatlab.cmake* in *cmake/modules* and its application in *CMakeLists.txt*).

For information about the MEX wrapper source and the S-Function wrapper source themselves, please refer to the example section.

The TEB library *teb_package.lib* and the compiled MEX files are stored somewhere inside the build folder, depending on the platform used. Please check *build/bin/lib* and ∗/build/lib/∗ first.

Way 2: Compile MEX and S-Function wrapper using Matlab itself

Advantages:

- Avoids copying the MEX file from the build path to the working directory, since Matlab creates it directly in the current directory.
- Allows compiling with *Matlab's/Simulink's Realtime Workshop*

Disadvantages:

- Allows the library *teb_package.lib* to be compiled only in Release mode.
- A second independent build process needs to be specified that requires information about libraries, include directories and additional source files.

Choosing this way requires the user to define settings for compiling a MEX wrapper in Matlab. First make sure to have at least *Visual Studio 2013* installed. Check if you set up your compiler in Matlab correctly:

```
mex -setup
```

Now switch to your working directory containing the MEX wrapper source file (For the sake of simplicity, we assume that additional header files are stored in the same folder, otherwise add the corresponding path to build process).

Compile your source code using *mex '-IC:\path-to-include' '-LC:\path-to-library' '-lLibName' 'source-file-name.cpp'*. If you want to use variables, use the following syntax and exchange the strings with variables:

```
mex(['-I' 'C:\teb_package\include'],...           // add project include dir
    ['-I' 'C:\teb_package\extern\eigen3'],...     // add path to Eigen-lib headers
    ['-L' 'C:\teb_package\build\lib\Release'],... // path to the package library teb_package.lib
    ['-l' 'teb_package'],...                      // the filename of the lib (without extension)
    'integrator_system_mex.cpp');                 // Source file for the MEX or S-Function wrapper.
```

If everything was successful you should now see a compiled MEX-file in your working directory. Please refer to the example section for more details on how to use the MEX/S-Function interface.

Use your code in Simulink

Create your model and add the S-Function block where you can specify the name of your compiled MEX S-Function. You can use your model in *Simulink Normal Mode* without Compiling

Compile your code in Simulink

If you want to compile your code in Simulink, go to *Simulation/Model Configuration Paramters/Code Generation* and switch the Language to C++. Select the *Interface* subsection and deselect *MAT-file logging*. Finally, you need to provide all build informations that was required by the mex compiler. Goto the *Custom Code* subsection. In the fragment *Include list of additional* add all paths and files that are required (find here the integrator example):

- Include directories:

  C:\teb_package\include

  C:\teb_package\extern\eigen3

- Libraries:

  C:\teb_package\build\lib\Release\teb_package.lib

Instead of linking the *teb_package.lib*, you can add the C++ source files (e.g. to avoid *error LNK2038* (see below)) of the package directly:

- Source files: *C:\teb_package\src\base\base_controller.cpp*

  C:\teb_package\src\base\base_solver_least_squares.cpp

  C:\teb_package\src\base\workspaces.cpp

  C:\teb_package\src\solver\solver_levenbergmarquardt_eigen_dense.cpp

  C:\teb_package\src\solver\solver_levenbergmarquardt_eigen_sparse.cpp

  Now you can compile your code directly within Simulink. If you have any troubles, please check the *Troubleshooting* section first.

Compile your code for Simulink xPC Target

Go to *Simulation/Model Configuration Paramters/Code Generation* and change the target to *Simulink Real-Time*. Afterwards follow the steps mentioned in *Compile your code in Simulink* above. But use the source files method instead of linking the precompiled library *teb_package.lib* before.

Simulink xPC Target does not support IO. Therefore you have to define *RTW* before compiling the teb_package / model. Define it using the CXX Flag ∗-DRTW∗ or uncomment the line '#define RTW' in the file *typedefs.h*.

Now you can compile you should be able to compile your model for xPC target.

Compile your code for Simulink Real-Time Windows Target

Compilation for RTWIN is currently only supported in *Normal Mode*. Compiling in *External Mode* invokes a dedicated Real-Time Kernel that does not support dynamic memory allocation and exceptions. Those are both utilized in the C++ standard library (STL) which is widely used within this project.

Troubleshooting

- *error LNK2038: mismatch detected for '_ITERATOR_DEBUG_LEVEL': value '2' doesn't match value '0'*

  Try to compile the TEB package in Release mode: Check first lines in *CMakeLists.txt*!

- *error LNK2038: Konflikt ermittelt für "RuntimeLibrary": Der Wert "MD_DynamicRelease" stimmt nicht mit dem Wert "MT_StaticRelease" xxx_sfun.obj überein.*

  Simulink compiles determines that some parts are compiled as static and some other parts as dynamic, therefore the linker cannot link them. For me it remains unclear, since the teb_package.lib is compiled statically and the mex compiler is Visual Studio in both cases. You can bypass this error by modifying the compiler settings in *Simulation/Model Configuration Paramters/Code Generation* by changing the *Build configuration* to *Specify* and append the ∗/MD∗ flag to the C++ Compiler flags. If you have still problems, try to exchange /∗MD∗ with ∗/MT∗. See the image below: An alternative way to solve the problem is to directly compile the whole teb_package by providing the ∗.cpp source files instead of the library *teb_package.lib*.

Chapter 6

Test List

Class teb::SolverNloptPackage

This class needs more testing (different solvers etc.)

Member teb::TebPlotter::plotTwoCol (const Eigen::Ref< const Eigen::VectorXd > &col1_time, const Eigen::Ref< const Eigen::MatrixXd > &col1_data, const Eigen::Ref< const Eigen::VectorXd > &col2_time, const Eigen::Ref< const Eigen::MatrixXd > &col2_data, PlotOptions *options=nullptr)

What happens if the right column contains more rows than the left one? (Alignment)

Chapter 7

Todo List

Class teb::BaseController

Implement complete API (abstract functions, etc.) to allow a generic controller usage for teb::Simulator and teb::TebPlotter

Member teb::BaseEdge< D, FuncType >::allocateMemory (bool skip_hessian=false)

Implement automatic memory allocation instead calling this function manually.

Parameters

skip_hessian	If no hessian calculation is required by the solver skip memory allocation.

Member teb::BaseEdge< D, FuncType >::calculateVertexDimensions ()

Maybe implement another strategy to store the dimensions.

Member teb::BaseEdge< D, FuncType, Vertices >::allocateMemory (bool skip_hessian=false)

Implement automatic memory allocation instead calling this function manually.

Parameters

skip_hessian	If no hessian calculation is required by the solver skip memory allocation.

Member teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ()

Maybe implement static/constexpr version of collecting all dimensions since they are nown at compile-time (maybe `http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array-in`

Member teb::BaseSolver::solve (HyperGraph ∗optimizable_graph)

Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

Member teb::BaseSolverLeastSquares::getChi2 () const

Maybe switch to the formulation $f(\mathscr{B}) = \mathbf{f}(\mathscr{B})^T \Omega \mathbf{f}(\mathscr{B})$ similar to g2o and Teb-Matlab.

Class teb::BaseSolverNonlinearProgramDense

The documentation of this class, after this class passed a couple of more tests

Class teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >

Maybe use a special representation or zero Hessians to avoid processing a lot of zeros (same for other edges)

Class teb::EdgeQuadraticForm< p, q >

Currently the Levenberg-Marquardt solver take the cost function squared by itself, therefore we need to implement the sqrt here: For other solvers this cost function will be problematic without taking the square.

Class teb::HessianWorkspace

TebVertex::dimension() is used at the moment. Acutally only a number of TebVertex::dimensionFree() non-zeros may appear. Test if it speeds up the optimization to skip those values by using the fixed-mask. But for common MPC problems only the first or/and the final state are fixed partially.

Class teb::JacobianWorkspace

TebVertex::dimension() is used at the moment. Acutally only a number of TebVertex::dimensionFree() non-zeros may appear. Test if it speeds up the optimization to skip those values by using the fixed-mask. But for common MPC problems only the first or/and the final state are fixed partially.

Class teb::NumericalIntegrator< p, q >

Implement and test Runge Kutta with derivatives http://www.emis.de/journals/EJDE/conf-proc/02/g1/goeke pdf

Class teb::PlotOptions

Extend functionality

Class teb::Simulator< p, q >

Add disturbance models.

Add reference trajectory or multiple reference steps for closed-loop control.

Member teb::Simulator< p, q >::setSampleTime (double sample_time)

Allow different sample_times for controller and plant simulation (the TEB sample time ΔT might be higher ...)

Class teb::SolverLevenbergMarquardtEigenSparse

Consider only the upper diagonal part of the hessian?

Member teb::SolverLevenbergMarquardtEigenSparse::buildJacobian ()

Check if implemented ordering of column and row iterations fits Eigen's column major representation best.

Class teb::SolverSQPDense

The documentation of this class, after the SQP sovler passed a couple of remaining (robustness) tests

Class teb::SystemDynamics< p, q >

Methods/Inteface for jacobian and hessian calculation

Class teb::TebController< p, q >

Using the TebController inside a standard container (e.g. vector) does not work as supposed to do.

Create copy and move constructors.

Member teb::TebController< p, q >::activateControlBounds (unsigned int control_idx, double u_min, double u_max, bool activate=true)

Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

Member teb::TebController< p, q >::activateStateBounds (unsigned int state_idx, double x_min, double x_max, bool activate=true)

Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

Member teb::TebController< p, q >::activateStateBounds (const Eigen::Ref< const StateVector > &x_min, const Eigen::Ref< const StateVector > &x_max, bool activate=true)

Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

Member teb::TebController< p, q >::buildOptimizationGraph ()

Here we need to fix the last control input since it undefined (no control is needed to go to state n+1). Maybe we can change the code somewhere else, that the contorl sequence is always getN()-1.

Member teb::TebController< p, q >::firstControlSaturated () const

Support multiple control input bounds (now its scalar for all elements of $\mathbf{u}$).

Returns

Saturated control input u_k [q x 1].

Member teb::TebController< p, q >::getActiveVertices ()

Add case for different state and control input sequence lengths (e.g. for move blocking)

Member teb::TebController< p, q >::initTrajectory (const Eigen::Ref< const StateVector > &x0, const Eigen::Ref< const StateVector > &xf, unsigned int n)

More efficient implementation that keeps previously allocated memory

Member teb::TebController< p, q >::optimizeTEB ()

The current implementation is really inefficient, since in each outer-loop-iteration all hyper-graph edges are deleted and afterwards reinstantiated. Create a dedicated update function in the future to hot-start from a previous hyper-graph.

Member teb::TebController< p, q >::resampleTrajectory (unsigned int n_new)

More efficient strategy without copying the complete sequences.

: later this should be n_new-1

Member teb::TebController< p, q >::returnControlInputSequence () const

Here we assume, that the last control input is invalid - We need to change this, if we have different sizes for state and control input sequences

Chapter 8

Bug List

Member teb::Config::Optim::Solver::Lsq::soft_constr_epsilon

Do not use at the moment, because it is not implemented correctly

Class teb::EdgeSystemDynamics< p, q, central >

Central differences do not seem to work as expected at the moment.

Member teb::Simulator< p, q >::simClosedLoop (const double *const x0, const double *const xf, double sim_time, bool manual_stepping=false, bool plot_result=true, bool plot_steps=true, BaseController *ctrl=nullptr)

If we set a title or legend in the plot options for stepping (!), than the plot won't be rendered correctly.

Class teb::TebPlotter

EPS and PDF export are not working as expected. No text is displayed and after importing to Inskape, the plot is blank (noticed on Ubuntu 14.04).

Chapter 9

Module Index

9.1 Modules

Here is a list of all modules:

Chapter 10

Namespace Index

10.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 11

Hierarchical Index

11.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 12

Class Index

12.1 Class List

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

Chapter 13

File Index

13.1 File List

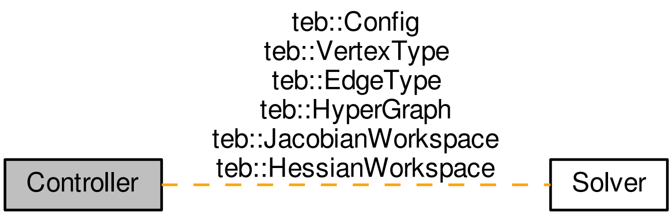
Here is a list of all files with brief descriptions:

Chapter 14

Module Documentation

14.1 Controller

Collaboration diagram for Controller:



Classes

- class teb::BaseController

  Base controller class (General Controller API)
- class teb::BaseEdge< D, FuncType, Vertices >

  Templated base edge class that stores an arbitary number of values.
- class teb::BaseEdge< D, FuncType >

  Templated base edge class that stores an arbitary number of values (Partial template specialization that allows changing vertices dimensions at runtime).
- class teb::EdgeSystemDynamics< p, q, central >

  Equality constraint edge for satisfying continious system dynamics specified by an SystemDynamics object.
- class teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >

  This class captures general bound constraints on optimization variables / vertices.
- class teb::EdgeMinimizeTime

  Objective edge: minimize ΔT.
- class teb::EdgeQuadraticForm< p, q >

  Objective edge: quadratic form for states and control input.
- class teb::Config

  Configurations for Controller and Solver classes.

- class teb::VertexType

  Generic interface class for vertices.
- class teb::EdgeType

  Generic interface class for edges.
- class teb::HyperGraph

  Graph object that stores pointers to active vertices and edges.
- class teb::SystemDynamics< p, q >

  Helper class for modeling nonlinear dynamic systems.
- class teb::TebController< p, q >

  Main Timed-Elastic-Band controller class.
- class teb::StateVertex< p >

  Vertex that contains the StateVector.
- class teb::ControlVertex< q >

  Vertex that contains the ControlVector.
- class teb::TimeDiff

  Vertex that contains the time information of the TEB: ΔT.
- class teb::JacobianWorkspace

  Memory management and accessor functions for the Block-Jacobians (for each edge).
- class teb::HessianWorkspace

  Memory management and accessor functions for the Block-Hessian (for each edge).

Typedefs

- using teb::EdgePositiveTime = BoundConstraint< BOUND_TYPE::LOWER, TimeDiff, BOUND_VARS::SINGLE, 0 >

  Bound constraint edge for the time difference: $\Delta T \geq \varepsilon$.
- template<int q>
  using teb::EdgeControlBounds = BoundConstraint< BOUND_TYPE::LOWERUPPER, ControlVertex< q >, BOUND_VARS::ALL >

  Bound constraint edge for control inputs: $\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}$.
- template<int p>
  using teb::EdgeStateBounds = BoundConstraint< BOUND_TYPE::LOWERUPPER, StateVertex< p >, BOUND_VARS::ALL >

  Bound constraint edge for states: $\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}$.

14.1.1 Detailed Description

14.1.2 Typedef Documentation

14.1.2.1 template<int q> using **teb::EdgeControlBounds** = typedef BoundConstraint<BOUND_TYPE::LOWERUPPER, ControlVertex<q>, BOUND_VARS::ALL>

The following two inequality constraints are implemented:

$$\mathbf{u}_k \geq \mathbf{u}_{min} \tag{14.1}$$
$$\mathbf{u}_k \leq \mathbf{u}_{max} \tag{14.2}$$

$\mathbf{u}_{min}$ and $\mathbf{u}_{max}$ are the lower and upper bounds on the control input $\mathbf{u}_k$.

Vertices required (connect the edge with the following vertices):

1. ControlVertex $\mathbf{u}_k$

Template Parameters

p	Number of state variables
q	Number of input variables

Definition at line 111 of file common_teb_edges.h.

14.1.2.2 using **teb::EdgePositiveTime** = typedef BoundConstraint<BOUND_TYPE::LOWER, TimeDiff, BOUND_VARS::SINGLE, 0>

The following inequality constraint is implemented:

$$\Delta T \geq \varepsilon$$

ε denotes an offset.

Set the offset ε using setBounds().

Vertices required (connect the edge with the following vertices):

1. TimeDiff ΔT

Definition at line 89 of file common_teb_edges.h.

14.1.2.3 template<int p> using **teb::EdgeStateBounds** = typedef BoundConstraint<BOUND_TYPE::LOWERUPPER, StateVertex<p>, BOUND_VARS::ALL>

The following two inequality constraints are implemented:

$$\mathbf{x}_k \geq \mathbf{x}_{min} \tag{14.3}$$

$$\mathbf{x}_k \leq \mathbf{x}_{max} \tag{14.4}$$

$\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ are the lower and upper bounds on the state vector $\mathbf{x}_k$.

Vertices required (connect the edge with the following vertices):

1. StateVertex $\mathbf{x}_k$

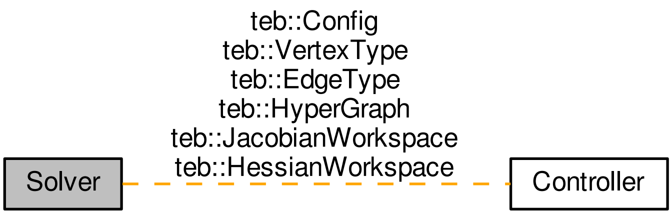
Template Parameters

p	Number of state variables
q	Number of input variables

Definition at line 133 of file common_teb_edges.h.

14.2 Solver

Collaboration diagram for Solver:



Classes

- class teb::BaseSolver

  Base class for solver implementations.
- class teb::BaseSolverLeastSquares

  Extended base solver class for least squares optimizations.
- class teb::Config

  Configurations for Controller and Solver classes.
- class teb::VertexType

  Generic interface class for vertices.
- class teb::EdgeType

  Generic interface class for edges.
- class teb::HyperGraph

  Graph object that stores pointers to active vertices and edges.
- class teb::JacobianWorkspace

  Memory management and accessor functions for the Block-Jacobians (for each edge).
- class teb::HessianWorkspace

  Memory management and accessor functions for the Block-Hessian (for each edge).
- class teb::BaseSolverNonlinearProgramDense

  Extended base solver class for nonlinear programs (dense version).
- class teb::SolverLevenbergMarquardtEigenDense

  Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Dense matrices version).
- class teb::SolverLevenbergMarquardtEigenSparse

  Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Sparse matrices version).
- class teb::SolverNloptPackage

  This class wraps the optimization problem to allow solving by NLOPT.
- class teb::SolverSQPDense

  Dense sequential quadratic programming (SQP) solver for nonlinear programs.

14.2.1 Detailed Description

14.3 Simulation

Classes

- class teb::NumericalIntegrator< p, q >

  Interface for numerical integration methods used in simulation.
- class teb::ExplicitEuler< p, q >

  Simple explicit euler method for integration.
- class teb::RungeKuttaClassic< p, q >

  Classic Runge Kutta method (fourth order)
- class teb::RungeKutta5thOrder< p, q >

  Runge Kutta method (fifth order, slighly modified)
- class teb::SimResults

  Simulation results are stored and combined in this container class.
- class teb::Simulator< p, q >

  Simulator for dynamic systems controlled by a TebController.

14.3.1 Detailed Description

14.4 Visualization

Classes

- struct teb::PlotOptions

  Customize plots and figures.
- class teb::TebPlotter

  Provides useful visualization and plotting functions.

14.4.1 Detailed Description

Chapter 15

Namespace Documentation

15.1 teb Namespace Reference

General project namespace for all library functions and objects.

Classes

- class BaseController

  Base controller class (General Controller API)
- class BaseEdge

  Templated base edge class that stores an arbitary number of values.
- class BaseEdge< D, FuncType >

  Templated base edge class that stores an arbitary number of values (Partial template specialization that allows changing vertices dimensions at runtime).
- class EdgeSystemDynamics

  Equality constraint edge for satisfying continious system dynamics specified by an SystemDynamics object.
- class BaseSolver

  Base class for solver implementations.
- class BaseSolverLeastSquares

  Extended base solver class for least squares optimizations.
- struct CustomBoundData

  Datastruct for custom bound data that can be queried from the BoundConstraint class.
- class BoundConstraint

  This class captures general bound constraints on optimization variables / vertices.
- class EdgeMinimizeTime

  Objective edge: minimize ΔT.
- class EdgeQuadraticForm

  Objective edge: quadratic form for states and control input.
- class Config

  Configurations for Controller and Solver classes.
- class VertexType

  Generic interface class for vertices.
- class EdgeType

  Generic interface class for edges.
- class HyperGraph

  Graph object that stores pointers to active vertices and edges.
- class SystemDynamics

Helper class for modeling nonlinear dynamic systems.

- class TebController

  Main Timed-Elastic-Band controller class.

- class StateVertex

  Vertex that contains the StateVector.

- class ControlVertex

  Vertex that contains the ControlVector.

- class TimeDiff

  Vertex that contains the time information of the TEB: ΔT.

- class JacobianWorkspace

  Memory management and accessor functions for the Block-Jacobians (for each edge).

- class HessianWorkspace

  Memory management and accessor functions for the Block-Hessian (for each edge).

- class NumericalIntegrator

  Interface for numerical integration methods used in simulation.

- class ExplicitEuler

  Simple explicit euler method for integration.

- class RungeKuttaClassic

  Classic Runge Kutta method (fourth order)

- class RungeKutta5thOrder

  Runge Kutta method (fifth order, slighly modified)

- class SimResults

  Simulation results are stored and combined in this container class.

- class Simulator

  Simulator for dynamic systems controlled by a TebController.

- class BaseSolverNonlinearProgramDense

  Extended base solver class for nonlinear programs (dense version).

- class SolverLevenbergMarquardtEigenDense

  Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Dense matrices version).

- class SolverLevenbergMarquardtEigenSparse

  Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Sparse matrices version).

- class SolverNloptPackage

  This class wraps the optimization problem to allow solving by NLOPT.

- class SolverSQPDense

  Dense sequential quadratic programming (SQP) solver for nonlinear programs.

- struct PlotOptions

  Customize plots and figures.

- class TebPlotter

  Provides useful visualization and plotting functions.

Typedefs

- using EdgePositiveTime = BoundConstraint< BOUND_TYPE::LOWER, TimeDiff, BOUND_VARS::SINGLE, 0 >

  Bound constraint edge for the time difference: $\Delta T \geq \varepsilon$.

- template<int q>
  using EdgeControlBounds = BoundConstraint< BOUND_TYPE::LOWERUPPER, ControlVertex< q >, BOUND_VARS::ALL >

  Bound constraint edge for control inputs: $\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}$.

- template<int p>
  using EdgeStateBounds = BoundConstraint< BOUND_TYPE::LOWERUPPER, StateVertex< p >, BOUND_VARS::ALL >

  Bound constraint edge for states: $\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}$.
- template<typename T >
  using EigenScalar = Eigen::Matrix< T, 1, 1 >

  Define 1x1 Eigen::Matrix.
- using EigenScalarD = EigenScalar< double >

  Define 1x1 Eigen::Matrix of type double.
- template<typename T >
  using BackupStackType = std::stack< T, std::deque< T > >

  Backup stack type.

Enumerations

- enum BOUND_TYPE { BOUND_TYPE::LOWER, BOUND_TYPE::UPPER, BOUND_TYPE::LOWERUPPER }

  Enumerator class that stores all types of bounds for consideration in BoundConstraint class.
- enum BOUND_VARS { BOUND_VARS::ALL, BOUND_VARS::SINGLE }

  Enumerator class that stores all types of variables that can be bounded using BoundConstraint class.
- enum FiniteDifferences { FiniteDifferences::FORWARD, FiniteDifferences::CENTRAL }

  Enumeration for different finite difference types used especially for dynamic system discretization.
- enum HessianMethod {
  HessianMethod::NUMERIC, HessianMethod::BLOCK_BFGS, HessianMethod::FULL_BFGS, HessianMethod::FULL_BFGS_WITH_STRUCTURE_FILTER,
  HessianMethod::ZERO_HESSIAN }

  Enumeration for different hessian calculation methods (not necessary for least-squares solver)
- enum HessianInit { HessianInit::ZERO, HessianInit::IDENTITY, HessianInit::NUMERIC }

  Enumeration for the hessian initialization (relevant for BFGS hessian approximations)
- enum NloptAlgorithms { NloptAlgorithms::SLSQP }

  Enumeration for different solver algorithms supported by the Nlopt library (only those that are working here)
- enum FUNCT_TYPE {
  FUNCT_TYPE::NONLINEAR, FUNCT_TYPE::NONLINEAR_SQUARED, FUNCT_TYPE::NONLINEAR_ONCE_DIFF, FUNCT_TYPE::QUADRATIC,
  FUNCT_TYPE::LINEAR_SQUARED, FUNCT_TYPE::LINEAR }

  Enum for different function types (see EdgeType class)
- enum NumericalIntegrators { NumericalIntegrators::EXPLICIT_EULER, NumericalIntegrators::RUNGE_KUTTA_CLASSIC, NumericalIntegrators::RUNGE_KUTTA_5TH }

  Enumeration for different numerical integration methods.

15.1.1 Typedef Documentation

15.1.1.1 template<typename T > using **teb::BackupStackType** = typedef std::stack<T, std::deque<T> >

Definition at line 52 of file typedefs.h.

15.1.1.2 template<typename T > using **teb::EigenScalar** = typedef Eigen::Matrix<T,1,1>

Definition at line 45 of file typedefs.h.

15.1.1.3 using teb::EigenScalarD = typedef EigenScalar<double>

Definition at line 48 of file typedefs.h.

15.1.2 Enumeration Type Documentation

15.1.2.1 enum teb::BOUND_TYPE [strong]

Enumerator

LOWER
UPPER
LOWERUPPER

Definition at line 14 of file bound_constraints.h.

15.1.2.2 enum teb::BOUND_VARS [strong]

Enumerator

ALL
SINGLE

Definition at line 16 of file bound_constraints.h.

15.1.2.3 enum teb::FiniteDifferences [strong]

Enumerator

FORWARD Forward differences: $\dot{x}(t) = \frac{x_{k+1} - x_k}{\Delta T}$.
CENTRAL Central differences: $\dot{x}(t) = \frac{x_{k+1} - x_{k-1}}{2\Delta T}$.

Definition at line 11 of file config.h.

15.1.2.4 enum teb::FUNCT_TYPE [strong]

Enumerator

NONLINEAR General nonlinear function that is at least two times differentiable.
NONLINEAR_SQUARED General nonlinear function that will be squared within the solver.
NONLINEAR_ONCE_DIFF Nonlinear function that is only differentiable once: the hessian is zero.
QUADRATIC Quadratic function (hessian is constant)
LINEAR_SQUARED Linear function that will be squared by the compiler.
LINEAR Linear function (hessian is zero)

Definition at line 79 of file graph.h.

15.1.2.5 enum teb::HessianInit [strong]

Enumerator

ZERO Use the zero matrix for initialization.
IDENTITY Use the identity matrix for hessian initialization (scale it via seperate config parameter)
NUMERIC Calculate the initialization of the hessian numerically using the original problem.

Definition at line 29 of file config.h.

15.1.2.6 enum teb::HessianMethod [strong]

Enumerator

NUMERIC Calculate hessian numerically via difference quotients (based on graph structure)

BLOCK_BFGS Calculate hessian for each block (defined by the graph) via BFGS.

FULL_BFGS Calculate the hessian via BFGS quasi-netwon method without considering strucutre.

FULL_BFGS_WITH_STRUCTURE_FILTER Calculate hessian like with FULL_BFGS but consider structure for non-zeros.

ZERO_HESSIAN Set Hessian to zero.

Definition at line 19 of file config.h.

15.1.2.7 enum teb::NloptAlgorithms [strong]

Enumerator

SLSQP Sequential quadratic programming approach.

Definition at line 38 of file config.h.

15.1.2.8 enum teb::NumericalIntegrators [strong]

Enumerator

EXPLICIT_EULER Forward Euler.

See Also

ExplicitEuler

RUNGE_KUTTA_CLASSIC Runge Kutta 4th Order.

See Also

RungeKuttaClassic

RUNGE_KUTTA_5TH Runge Kutta 5th Order.

See Also

RungeKutta5thOrder

Definition at line 12 of file integrators.h.

Chapter 16

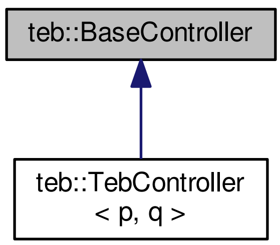
Class Documentation

16.1 teb::BaseController Class Reference

Base controller class (General Controller API)

```
#include <base_controller.h>
```

Inheritance diagram for teb::BaseController:



Public Member Functions

- virtual double getDt () const =0

  Get the time difference ΔT between two discrete samples within the horizon.
- virtual unsigned int getN () const =0

  Get the horizon length, resp.
- virtual void step (const double *const x0, const double *const xf, double *ctrl_out=nullptr)=0

  Perform complete MPC step (initialization if necessary or update and optimization)
- virtual Eigen::MatrixXd getStateCtrlInfoMat () const =0

  Get a matrix containing all state and control input sequences.
- virtual Eigen::VectorXd getAbsoluteTimeVec () const =0

  Get the vector of absolute times for the complete horizon.
- virtual Eigen::VectorXd firstState () const =0

  Access the first state $\mathbf{x}_0$ of the horizon.
- virtual Eigen::VectorXd lastState () const =0

Access the last state $\mathbf{x}_n$ of the horizon.
- virtual Eigen::VectorXd firstControl () const =0

  Access the first control input $\mathbf{u}_1$ of the horizon.
- virtual Eigen::MatrixXd returnControlInputSequence () const =0

  Return the complete control input sequence $u_k, k = 1, \ldots, n-1$.
- virtual void resetController ()=0

  Reset the controller.

16.1.1 Detailed Description

This abstract class defines the general controller interface.

Todo Implement complete API (abstract functions, etc.) to allow a generic controller usage for teb::Simulator and teb::TebPlotter

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

teb::TebController

Definition at line 27 of file base_controller.h.

16.1.2 Member Function Documentation

16.1.2.1 virtual Eigen::VectorXd teb::BaseController::firstControl ( ) const [pure virtual]

Returns

Copy of the first state

Implemented in teb::TebController< p, q >.

Referenced by teb::Simulator< p, q >::simClosedLoop().

16.1.2.2 virtual Eigen::VectorXd teb::BaseController::firstState ( ) const [pure virtual]

Returns

Copy of the first state

Implemented in teb::TebController< p, q >.

Referenced by teb::Simulator< p, q >::simOpenLoop().

16.1.2.3 virtual Eigen::VectorXd teb::BaseController::getAbsoluteTimeVec ( ) const [pure virtual]

In case of a uniform step width / time difference (see getDt()) the vector corresponds to $t = [0, \Delta T, 2\Delta T, \ldots, n\Delta T]$.

Returns

vector of absolute times starting with t=0s.Get vector of absolute times $t = [0, \Delta T, 2\Delta T, \ldots, n\Delta T]$.

Implemented in teb::TebController< p, q >.

Referenced by teb::Simulator< p, q >::simOpenLoop().

16.1.2.4 virtual double teb::BaseController::getDt ( ) const [pure virtual]

It can also be interpreted as the step width.

Returns

ΔT

Implemented in teb::TebController< p, q >.

Referenced by teb::Simulator< p, q >::simClosedLoop(), and teb::Simulator< p, q >::simOpenLoop().

16.1.2.5 virtual unsigned int teb::BaseController::getN ( ) const [pure virtual]

the number of discrete samples.

Returns

Number of discrete samples

Implemented in teb::TebController< p, q >.

Referenced by teb::Simulator< p, q >::simOpenLoop().

16.1.2.6 virtual Eigen::MatrixXd teb::BaseController::getStateCtrlInfoMat ( ) const [pure virtual]

The purpose of this function is mainly for debugging and visualization stuff.

Returns

[p+q x n] matrix with `p` states, `q` control inputs and n=getN().

Implemented in teb::TebController< p, q >.

Referenced by teb::Simulator< p, q >::simOpenLoop().

16.1.2.7 virtual Eigen::VectorXd teb::BaseController::lastState ( ) const [pure virtual]

Returns

Copy of the last state

Implemented in teb::TebController< p, q >.

16.1.2.8 virtual void teb::BaseController::resetController ( ) [pure virtual]

Implemented in teb::TebController< p, q >.

Referenced by teb::Simulator< p, q >::simClosedLoop(), and teb::Simulator< p, q >::simOpenLoop().

16.1.2.9 virtual Eigen::MatrixXd teb::BaseController::returnControlInputSequence ( ) const [pure virtual]

Returns

Matrix containing all planned control inputs [q x n-1]

Implemented in teb::TebController< p, q >.

Referenced by teb::Simulator< p, q >::simOpenLoop().

16.1.2.10 virtual void teb::BaseController::step (const double ∗const *x0*, const double ∗const *xf*, double ∗ *ctrl_out* = nullptr) [pure virtual]

Parameters

x0	double array containing start state `x0` with `p` components [`p` x 1]
xf	double array containing final state `xf` with `p` components [`p` x 1]
ctrl_out	[output] store control input (firstControl()) to control the plant [`q` x 1]

Implemented in teb::TebController< p, q >.

Referenced by teb::Simulator< p, q >::simClosedLoop(), and teb::Simulator< p, q >::simOpenLoop().

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

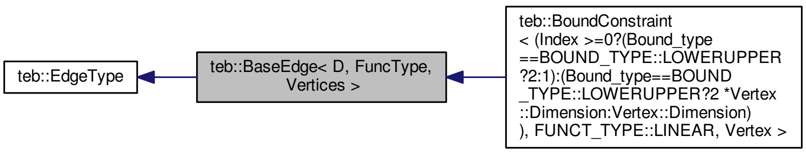
- base_controller.h

16.2 teb::BaseEdge< D, FuncType, Vertices > Class Template Reference

Templated base edge class that stores an arbitary number of values.

`#include <base_edge.h>`

Inheritance diagram for teb::BaseEdge< D, FuncType, Vertices >:



Public Types

- using ValueVector = Eigen::Matrix< double, D, 1 >

    Typedef to represent the static Eigen::Vector of cost/constraint values.
- using EdgeContainer = std::vector< EdgeType ∗ >

    Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).
- using ValueVectorMap = Eigen::Map< Eigen::VectorXd >

    Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.

Public Member Functions

- virtual int dimension () const

    Return edge dimension by function call.
- BaseEdge ()=delete

    Delete default constructor.
- template<class... VerticesT>
  BaseEdge (VerticesT &...args)
- virtual ∼BaseEdge ()

    < Construct edge by passing all vertices references.
- virtual FUNCT_TYPE functType () const

    Return function typed given by template parameter FuncType.
- virtual const double ∗ valuesData () const

    Return pointer to cost/constraint values [dimension() x 1] (read-only).

- virtual double $*$ valuesData ()

  Return pointer to cost/constraint values [dimension() x 1].
- virtual const ValueVector & values () const

  Return cost/constraint values as static Eigen::Vector (read-only).
- virtual ValueVector & values ()

  Return cost/constraint values as static Eigen::Vector.
- virtual void allocateMemory (bool skip_hessian=false)

  Allocate memory for JacobianWorkspace and HessianWorkspace and get vertices dimensions.
- virtual VertexType $*$ getVertex (unsigned int idx)

  Access attached vertex with index `idx`.
- virtual const VertexType $*$ getVertex (unsigned int idx) const

  Access attached vertex with index `idx` (read-only).
- virtual unsigned int noVertices () const

  Return the number of attached vertices.
- virtual void calculateVertexDimensions ()

  Return the number of attached vertices.
- virtual const std::array
  $<$ VertexType $*$, NoVertices $>$ & vertices () const

  Return vertex container of the edge.
- virtual void computeValues ()

  Actual cost function or constraint function. For constraints it is computeValues() $<=$ 0.
- virtual void computeJacobian ()

  Actual cost function or constraint function. For constraints it is computeValues() $<=$ 0.
- virtual void computeHessian ()

  Compute Block-Hessian and store it to EdgeType::_hessians according to the description in HessianWorkspace.
- virtual const bool isBoundConstraint () const

  Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.
- virtual ValueVectorMap valuesMap ()

  Get cost/constraints values as Eigen type matrix.
- virtual void $*$ getCustomData ()

  Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.
- virtual JacobianWorkspace & jacobians ()

  Access the jacobian workspace of this edge.
- virtual const JacobianWorkspace & jacobians () const

  Access the jacobian workspace of this edge (read-only).
- virtual HessianWorkspace & hessians ()

  Access the hessian workspace of this edge.
- virtual const HessianWorkspace & hessians () const

  Access the jacobian workspace of this edge (read-only).
- virtual void backupJacobian ()

  Make a backup of the current jacobian block matrix.
- virtual void restoreJacobian ()

  Restore jacobian block matrix from the backup stack.
- virtual void discardJacobianBackup ()

  Discard current jacobian backup.
- virtual JacobianWorkspace & getJacobianBackup ()

Static Public Attributes

- static constexpr const int NoVertices = sizeof...(Vertices)
- static const int Dimension = D

    Edge dimension as static member variable.

Protected Attributes

- ValueVector _values = ValueVector::Zero()

    Actual cost/constraint value data object (initialized to zero).
- const std::array< VertexType
  ∗, NoVertices > _vertices

    Container that stores all attached vertices.
- std::array< int, NoVertices > _vertices_dim

    Store dimensions for all vertices (EdgeType::_vertices, calculateVertexDimensions())
- JacobianWorkspace _jacobians

    Block-Jacobians of the edge (see JacobianWorkspace).
- HessianWorkspace _hessians

    Block-Hessians of the edge (see HessianWorkspace)
- BackupStackType
  < JacobianWorkspace > _jacob_backup

16.2.1 Detailed Description

template<int D, FUNCT_TYPE FuncType, class... Vertices>class teb::BaseEdge< D, FuncType, Vertices >

This class defines the basis for nearly all edges. The dimension has to be known at compile time and is set by the template parameter `D`.

Remarks

Call allocateMemory() before using the edge for optimization.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

EdgeType

Template Parameters

D	Dimension of the edge (value vector).
FuncType	Function type of the underlying cost function (according to FUNC_TYPE enum)
Vertices...	An arbitary number of vertex types that are attached to this edge

Definition at line 41 of file base_edge.h.

16.2.2 Member Typedef Documentation

16.2.2.1 using teb::EdgeType::EdgeContainer = std::vector<EdgeType∗> [inherited]

Definition at line 114 of file graph.h.

16.2.2.2 template<int D, FUNCT_TYPE FuncType, class... Vertices> using teb::BaseEdge< D, FuncType, Vertices >::ValueVector = Eigen::Matrix<double, D, 1>

Definition at line 54 of file base_edge.h.

16.2.2.3 using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd > [inherited]

Definition at line 116 of file graph.h.

16.2.3 Constructor & Destructor Documentation

16.2.3.1 template<int D, FUNCT_TYPE FuncType, class... Vertices> teb::BaseEdge< D, FuncType, Vertices >::BaseEdge ( ) [delete]

16.2.3.2 template<int D, FUNCT_TYPE FuncType, class... Vertices> template<class... VerticesT> teb::BaseEdge< D, FuncType, Vertices >::BaseEdge (VerticesT &... *args*) [inline]

Definition at line 60 of file base_edge.h.

16.2.3.3 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual teb::BaseEdge< D, FuncType, Vertices >::~BaseEdge ( ) [inline], [virtual]

Empty Destructor

Definition at line 66 of file base_edge.h.

16.2.4 Member Function Documentation

16.2.4.1 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices >::allocateMemory (bool *skip_hessian* = false) [inline], [virtual]

Remarks

Before calling, call resizeVertexContainer() edge and add all vertices (setVertex()) first.

Todo Implement automatic memory allocation instead calling this function manually.

Parameters

skip_hessian	If no hessian calculation is required by the solver skip memory allocation.

Implements teb::EdgeType.

Definition at line 84 of file base_edge.h.

16.2.4.2 virtual void teb::EdgeType::backupJacobian ( ) [inline], [virtual], [inherited]

Definition at line 162 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.2.4.3 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ( ) [inline], [virtual]

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container EdgeType::_vertices_dim. This function is called within BaseEdge::allocate-Memory().

Todo Maybe implement static/constexpr version of collecting all dimensions since they are nown at compile-time (maybe http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array-

Definition at line 110 of file base_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), and teb::BaseEdge< D, FuncType >::allocateMemory().

16.2.4.4 template<int D, FUNCT_TYPE FuncType, class... Vertices> void teb::BaseEdge< D, FuncType, Vertices >::computeHessian ( ) [virtual]

This implementation calculates the Block-Hessians for all statically allocated vertices and cost/constraint values numerically using forward differences.

For more information about the Hessian structure refer to HessianWorkspace.

If the FuncType template parameter is set to FUNCT_TYPE::LINEAR or FuncType == FUNCT_TYPE::NONLINEA-R_ONCE_DIFF the hessian is set to zero.

The results are stored to the class member BaseEdgeBinary::_hessians.

Feel free to reimplement this function in child classes in order to provide analytical Hessians.

Remarks

This function assumes that computeJacobian() was called before! It is used as reference for calculating the forward-differences.

See Also

computeJacobian(), HessianWorkspace

Implements teb::EdgeType.

Reimplemented in teb::EdgeQuadraticForm< p, q >, teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >, and teb::EdgeMinimizeTime.

Definition at line 165 of file base_edge.hpp.

References teb::VertexType::dimension(), teb::JacobianWorkspace::getWorkspace(), teb::VertexType::isFixedAll(), teb::LINEAR, teb::LINEAR_SQUARED, teb::NONLINEAR_ONCE_DIFF, teb::VertexType::plus(), teb::VertexType-::pop(), and teb::VertexType::push().

16.2.4.5 template<int D, FUNCT_TYPE FuncType, class... Vertices> void teb::BaseEdge< D, FuncType, Vertices >::computeJacobian ( ) [virtual]

This implementation calculates the Block-Jacobians for all statically allocated vertices numerically using central differences.

Compute Block-Jacobian and store it to EdgeType::_jacobians according to the description in JacobianWorkspace.

For more information about the Jacobian structure refer to JacobianWorkspace.

The results are stored to the class member BaseEdgeBinary::_jacobians.

The implementation is similar to [2].

Feel free to reimplement this function in child classes in order to provide analytical Jacobians.

See Also

computeHessian(), JacobianWorkspace

Implements teb::EdgeType.

Reimplemented in teb::EdgeQuadraticForm< p, q >, teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >, and teb::EdgeMinimizeTime.

Definition at line 24 of file base_edge.hpp.

References teb::VertexType::dimension(), teb::VertexType::isFixedAll(), teb::VertexType::plus(), teb::VertexType::pop(), and teb::VertexType::push().

Referenced by teb::BaseEdge< D, FuncType >::computeHessian().

16.2.4.6 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices >::computeValues ( ) [inline], [virtual]

Implements teb::EdgeType.

Reimplemented in teb::EdgeQuadraticForm< p, q >, teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >, teb::EdgeSystemDynamics< p, q, central >, and teb::EdgeMinimizeTime.

Definition at line 125 of file base_edge.h.

Referenced by teb::BaseEdge< D, FuncType >::computeJacobian().

16.2.4.7 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual int teb::BaseEdge< D, FuncType, Vertices >::dimension ( ) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 51 of file base_edge.h.

16.2.4.8 virtual void teb::EdgeType::discardJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 171 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.2.4.9 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual FUNCT_TYPE teb::BaseEdge< D, FuncType, Vertices >::functType ( ) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 69 of file base_edge.h.

16.2.4.10 virtual void* teb::EdgeType::getCustomData ( ) [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 154 of file graph.h.

16.2.4.11 virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 173 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.2.4.12 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual VertexType∗ teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) [inline], [virtual]

Implements teb::EdgeType.

Definition at line 98 of file base_edge.h.

16.2.4.13 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const VertexType∗ teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 99 of file base_edge.h.

16.2.4.14 virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline], [virtual], [inherited]

Definition at line 159 of file graph.h.

References teb::EdgeType::_hessians.

16.2.4.15 virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline], [virtual], [inherited]

Definition at line 160 of file graph.h.

References teb::EdgeType::_hessians.

16.2.4.16 virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 124 of file graph.h.

16.2.4.17 virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline], [virtual], [inherited]

Definition at line 157 of file graph.h.

References teb::EdgeType::_jacobians.

16.2.4.18 virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline], [virtual], [inherited]

Definition at line 158 of file graph.h.

References teb::EdgeType::_jacobians.

16.2.4.19 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual unsigned int teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 101 of file base_edge.h.

16.2.4.20 virtual void teb::EdgeType::restoreJacobian ( ) [inline], [virtual], [inherited]

Definition at line 165 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.2.4.21 template< int D, FUNCT_TYPE FuncType, class... Vertices > virtual const ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) const [inline], [virtual]

Definition at line 74 of file base_edge.h.

16.2.4.22 template< int D, FUNCT_TYPE FuncType, class... Vertices > virtual ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) [inline], [virtual]

Definition at line 75 of file base_edge.h.

16.2.4.23 template< int D, FUNCT_TYPE FuncType, class... Vertices > virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 72 of file base_edge.h.

16.2.4.24 template< int D, FUNCT_TYPE FuncType, class... Vertices > virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline], [virtual]

Implements teb::EdgeType.

Definition at line 73 of file base_edge.h.

16.2.4.25 virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline], [virtual], [inherited]

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References teb::EdgeType::dimension(), and teb::EdgeType::valuesData().

16.2.4.26 template< int D, FUNCT_TYPE FuncType, class... Vertices > virtual const std::array< VertexType*,NoVertices>& teb::BaseEdge< D, FuncType, Vertices >::vertices ( ) const [inline], [virtual]

Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file base_edge.h.

16.2.5 Member Data Documentation

16.2.5.1 HessianWorkspace teb::EdgeType::_hessians [protected], [inherited]

Definition at line 178 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeMinimizeTime::computeHessian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), and teb::EdgeType::hessians().

16.2.5.2 BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup [protected], [inherited]

Definition at line 180 of file graph.h.

Referenced by teb::EdgeType::backupJacobian(), teb::EdgeType::discardJacobianBackup(), teb::EdgeType::getJacobianBackup(), and teb::EdgeType::restoreJacobian().

16.2.5.3 JacobianWorkspace teb::EdgeType::_jacobians [protected], [inherited]

Definition at line 173 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeType::backupJacobian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::EdgeMinimizeTime::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::EdgeType::jacobians(), and teb::EdgeType::restoreJacobian().

16.2.5.4 template<int D, FUNCT_TYPE FuncType, class... Vertices> ValueVector teb::BaseEdge< D, FuncType, Vertices >::_values = ValueVector::Zero() [protected]

Definition at line 139 of file base_edge.h.

Referenced by teb::BaseEdge< D, FuncType >::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeValues(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::values(), teb::BaseEdge< D, FuncType >::values(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::valuesData(), and teb::BaseEdge< D, FuncType >::valuesData().

16.2.5.5 template<int D, FUNCT_TYPE FuncType, class... Vertices> const std::array<VertexType∗, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices [protected]

Definition at line 141 of file base_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::calculateVertexDimensions(), teb::BaseEdge< D, FuncType >::calculateVertexDimensions(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeValues(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::getVertex(), teb::BaseEdge< D, FuncType >::getVertex(), teb::BaseEdge< D, FuncType >::noVertices(), teb::BaseEdge< D, FuncType >::resizeVertexContainer(), teb::BaseEdge< D, FuncType >::setVertex(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::vertices(), and teb::BaseEdge< D, FuncType >::vertices().

16.2.5.6 template<int D, FUNCT_TYPE FuncType, class... Vertices> std::array<int, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim [protected]

Definition at line 142 of file base_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::calculateVertexDimensions(), and teb::BaseEdge< D, FuncType >::calculateVertexDimensions().

16.2.5.7 template<int D, FUNCT_TYPE FuncType, class... Vertices> const int teb::BaseEdge< D, FuncType, Vertices >::Dimension = D [static]

Definition at line 50 of file base_edge.h.

16.2.5.8 template< int D, FUNCT_TYPE FuncType, class... Vertices > constexpr const int teb::BaseEdge< D, FuncType, Vertices >::NoVertices = sizeof...(Vertices) [static]

Definition at line 48 of file base_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::noVertices().

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

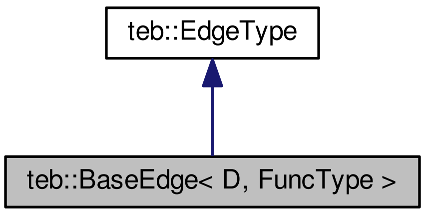
- base_edge.h
- base_edge.hpp

16.3 teb::BaseEdge< D, FuncType > Class Template Reference

Templated base edge class that stores an arbitary number of values (Partial template specialization that allows changing vertices dimensions at runtime).

`#include <base_edge.h>`

Inheritance diagram for teb::BaseEdge< D, FuncType >:



Public Types

- using ValueVector = Eigen::Matrix< double, D, 1 >

  Typedef to represent the static Eigen::Vector of cost/constraint values.
- using EdgeContainer = std::vector< EdgeType ∗ >

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).
- using ValueVectorMap = Eigen::Map< Eigen::VectorXd >

  Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.

Public Member Functions

- virtual int dimension () const

  Return edge dimension by function call.
- BaseEdge ()

  Empty Constructor.
- virtual ∼BaseEdge ()

  Empty Destructor.
- virtual FUNCT_TYPE functType () const

  Return function typed given by template parameter FuncType.

- virtual const double * valuesData () const

  Return pointer to cost/constraint values [dimension() x 1] (read-only).
- virtual double * valuesData ()

  Return pointer to cost/constraint values [dimension() x 1].
- virtual const ValueVector & values () const

  Return cost/constraint values as static Eigen::Vector (read-only).
- virtual ValueVector & values ()

  Return cost/constraint values as static Eigen::Vector.
- virtual void allocateMemory (bool skip_hessian=false)

  Allocate memory for JacobianWorkspace and HessianWorkspace and get vertices dimensions.
- virtual VertexType * getVertex (unsigned int idx)

  Access attached vertex with index `idx`.
- virtual const VertexType * getVertex (unsigned int idx) const

  Access attached vertex with index `idx` (read-only).
- virtual unsigned int noVertices () const

  Return the number of attached vertices.
- virtual void calculateVertexDimensions ()

  Return the number of attached vertices.
- virtual void setVertex (unsigned int idx, VertexType *pvertex)

  Attach a new vertex of type TebVertex with index `idx` to the edge.*
- virtual const std::vector
  < VertexType * > & vertices () const

  Return vertex container of the edge.
- virtual std::vector
  < VertexType * > & vertices ()

  Return vertex container of the edge.
- virtual void computeJacobian ()

  Compute Block-Jacobian and store it to EdgeType::_jacobians according to the description in JacobianWorkspace.
- virtual void computeHessian ()

  Compute Block-Hessian and store it to EdgeType::_hessians according to the description in HessianWorkspace.
- virtual const bool isBoundConstraint () const

  Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.
- virtual void computeValues ()=0

  Actual cost function or constraint function. For constraints it is computeValues() <= 0.
- virtual ValueVectorMap valuesMap ()

  Get cost/constraints values as Eigen type matrix.
- virtual void * getCustomData ()

  Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.
- virtual JacobianWorkspace & jacobians ()

  Access the jacobian workspace of this edge.
- virtual const JacobianWorkspace & jacobians () const

  Access the jacobian workspace of this edge (read-only).
- virtual HessianWorkspace & hessians ()

  Access the hessian workspace of this edge.
- virtual const HessianWorkspace & hessians () const

  Access the jacobian workspace of this edge (read-only).
- virtual void backupJacobian ()

  Make a backup of the current jacobian block matrix.
- virtual void restoreJacobian ()

Restore jacobian block matrix from the backup stack.
- virtual void discardJacobianBackup ()

  Discard current jacobian backup.
- virtual JacobianWorkspace & getJacobianBackup ()

Static Public Attributes

- static const int Dimension = D

  Edge dimension as static member variable.

Protected Member Functions

- void resizeVertexContainer (unsigned int n)

  Set number n of vertices attached to this edge.

Protected Attributes

- ValueVector _values = ValueVector::Zero()

  Actual cost/constraint value data object (initialized to zero).
- std::vector$<$ VertexType $* >$ _vertices

  Container that stores all attached vertices.
- std::vector$<$ int $>$ _vertices_dim

  Store dimensions for all vertices (EdgeType::_vertices, calculateVertexDimensions())
- JacobianWorkspace _jacobians

  Block-Jacobians of the edge (see JacobianWorkspace).
- HessianWorkspace _hessians

  Block-Hessians of the edge (see HessianWorkspace)
- BackupStackType
  $<$ JacobianWorkspace $>$ _jacob_backup

16.3.1 Detailed Description

template$<$int D, FUNCT_TYPE FuncType$>$class teb::BaseEdge$<$ D, FuncType $>$

This class defines the basis for nearly all edges. The dimension has to be known at compile time and is set by the template parameter `D`.

Remarks

Call allocateMemory() before using the edge for optimization.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

EdgeType

Template Parameters

D	Dimension of the edge (value vector).
FuncType	Function type of the underlying cost function (according to FUNC_TYPE enum)

Definition at line 172 of file base_edge.h.

16.3.2 Member Typedef Documentation

16.3.2.1 using teb::EdgeType::EdgeContainer = std::vector<EdgeType*> [inherited]

Definition at line 114 of file graph.h.

16.3.2.2 template<int D, FUNCT_TYPE FuncType> using teb::BaseEdge< D, FuncType >::ValueVector = Eigen::Matrix<double, D, 1>

Definition at line 180 of file base_edge.h.

16.3.2.3 using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd > [inherited]

Definition at line 116 of file graph.h.

16.3.3 Constructor & Destructor Documentation

16.3.3.1 template<int D, FUNCT_TYPE FuncType> teb::BaseEdge< D, FuncType >::BaseEdge ( ) [inline]

Definition at line 183 of file base_edge.h.

16.3.3.2 template<int D, FUNCT_TYPE FuncType> virtual teb::BaseEdge< D, FuncType >::~BaseEdge ( ) [inline], [virtual]

Definition at line 184 of file base_edge.h.

16.3.4 Member Function Documentation

16.3.4.1 template<int D, FUNCT_TYPE FuncType> virtual void teb::BaseEdge< D, FuncType >::allocateMemory (bool *skip_hessian* = false) [inline], [virtual]

Remarks

Before calling, call resizeVertexContainer() edge and add all vertices (setVertex()) first.

Todo Implement automatic memory allocation instead calling this function manually.

Parameters

skip_hessian	If no hessian calculation is required by the solver skip memory allocation.

Implements teb::EdgeType.

Definition at line 202 of file base_edge.h.

References teb::EdgeType::_hessians, teb::EdgeType::_jacobians, teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim, teb::JacobianWorkspace::allocate(), teb::HessianWorkspace::allocate(), and teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions().

16.3.4.2 virtual void teb::EdgeType::backupJacobian ( ) [inline], [virtual], [inherited]

Definition at line 162 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.3.4.3 template<int D, FUNCT_TYPE FuncType> virtual void teb::BaseEdge< D, FuncType >::calculateVertexDimensions ( ) [inline], [virtual]

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container EdgeType::_vertices_dim. This function is called within BaseEdge::allocate-Memory().

Todo Maybe implement another strategy to store the dimensions.

Definition at line 227 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices, and teb::BaseEdge< D, FuncType, Vertices >::-_vertices_dim.

16.3.4.4 template<int D, FUNCT_TYPE FuncType> void teb::BaseEdge< D, FuncType >::computeHessian ( ) [virtual]

This implementation calculates the Block-Hessians for all dynamically allocated vertices and cost/constraint values numerically using forward differences.

For more information about the Hessian structure refer to HessianWorkspace.

If the FuncType template parameter is set to FUNCT_TYPE::LINEAR or FuncType == FUNCT_TYPE::NONLINEA-R_ONCE_DIFF the hessian is set to zero.

The results are stored to the class member BaseEdgeBinary::_hessians.

Feel free to reimplement this function in child classes in order to provide analytical Hessians.

Remarks

> This function assumes that computeJacobian() was called before! It is used as reference for calculating the forward-differences.

See Also

> computeJacobian(), HessianWorkspace

Implements teb::EdgeType.

Definition at line 277 of file base_edge.hpp.

References teb::EdgeType::_hessians, teb::EdgeType::_jacobians, teb::BaseEdge< D, FuncType, Vertices >-::_vertices, teb::BaseEdge< D, FuncType, Vertices >::computeJacobian(), teb::VertexType::dimension(), teb-::JacobianWorkspace::getWorkspace(), teb::HessianWorkspace::getWorkspace(), teb::VertexType::isFixedAll(), teb::LINEAR, teb::LINEAR_SQUARED, teb::NONLINEAR_ONCE_DIFF, teb::VertexType::plus(), teb::VertexType-::pop(), and teb::VertexType::push().

16.3.4.5 template<int D, FUNCT_TYPE FuncType> void teb::BaseEdge< D, FuncType >::computeJacobian ( ) [virtual]

This implementation calculates the Block-Jacobians for all dynamically allocated vertices numerically using central differences.

For more information about the Jacobian structure refer to JacobianWorkspace.

The results are stored to the class member BaseEdgeBinary::_jacobians.

The implementation is similar to [2].

Feel free to reimplement this function in child classes in order to provide analytical Jacobians.

See Also

computeHessian(), JacobianWorkspace

Implements teb::EdgeType.

Definition at line 92 of file base_edge.hpp.

References teb::EdgeType::_jacobians, teb::BaseEdge< D, FuncType, Vertices >::_values, teb::BaseEdge< D, FuncType, Vertices >::_vertices, teb::BaseEdge< D, FuncType, Vertices >::computeValues(), teb::VertexType::dimension(), teb::JacobianWorkspace::getWorkspace(), teb::VertexType::isFixedAll(), teb::VertexType::plus(), teb::VertexType::pop(), and teb::VertexType::push().

16.3.4.6 virtual void teb::EdgeType::computeValues ( ) [pure virtual], [inherited]

Implemented in teb::EdgeQuadraticForm< p, q >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >, teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >, teb::EdgeSystemDynamics< p, q, central >, and teb::EdgeMinimizeTime.

16.3.4.7 template<int D, FUNCT_TYPE FuncType> virtual int teb::BaseEdge< D, FuncType >::dimension ( ) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 177 of file base_edge.h.

16.3.4.8 virtual void teb::EdgeType::discardJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 171 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.3.4.9 template<int D, FUNCT_TYPE FuncType> virtual FUNCT_TYPE teb::BaseEdge< D, FuncType >::functType ( ) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 187 of file base_edge.h.

16.3.4.10 virtual void* teb::EdgeType::getCustomData ( ) [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 154 of file graph.h.

16.3.4.11 virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 173 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.3.4.12 template< int D, FUNCT_TYPE FuncType > virtual VertexType* teb::BaseEdge< D, FuncType >::getVertex (unsigned int *idx*) [inline], [virtual]

Implements teb::EdgeType.

Definition at line 215 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.3.4.13 template< int D, FUNCT_TYPE FuncType > virtual const VertexType* teb::BaseEdge< D, FuncType >::getVertex (unsigned int *idx*) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 216 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.3.4.14 virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline], [virtual], [inherited]

Definition at line 159 of file graph.h.

References teb::EdgeType::_hessians.

16.3.4.15 virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline], [virtual], [inherited]

Definition at line 160 of file graph.h.

References teb::EdgeType::_hessians.

16.3.4.16 virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 124 of file graph.h.

16.3.4.17 virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline], [virtual], [inherited]

Definition at line 157 of file graph.h.

References teb::EdgeType::_jacobians.

16.3.4.18 virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline], [virtual], [inherited]

Definition at line 158 of file graph.h.

References teb::EdgeType::_jacobians.

16.3.4.19 template< int D, FUNCT_TYPE FuncType > virtual unsigned int teb::BaseEdge< D, FuncType >::noVertices ( ) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 218 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.3.4.20 template<int D, FUNCT_TYPE FuncType> void **teb::BaseEdge**< D, FuncType >::resizeVertexContainer (unsigned int *n*) [inline], [protected]

Definition at line 282 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.3.4.21 virtual void teb::EdgeType::restoreJacobian ( ) [inline], [virtual], [inherited]

Definition at line 165 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.3.4.22 template<int D, FUNCT_TYPE FuncType> virtual void **teb::BaseEdge**< D, FuncType >::setVertex (unsigned int *idx*, **VertexType** ∗ *pvertex*) [inline], [virtual]

Make sure that memory is allocated before using resizeVertexContainer() or that it is controlled by a specific child class.

Parameters

idx	index of the vertex inside this edge. Start with 0.
pvertex	pointer to the TebVertex object or to one of its children.

Definition at line 246 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.3.4.23 template<int D, FUNCT_TYPE FuncType> virtual const **ValueVector&** **teb::BaseEdge**< D, FuncType >::values ( ) const [inline], [virtual]

Definition at line 192 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_values.

16.3.4.24 template<int D, FUNCT_TYPE FuncType> virtual **ValueVector&** **teb::BaseEdge**< D, FuncType >::values ( ) [inline], [virtual]

Definition at line 193 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_values.

16.3.4.25 template<int D, FUNCT_TYPE FuncType> virtual const double∗ **teb::BaseEdge**< D, FuncType >::valuesData ( ) const [inline], [virtual]

Implements teb::EdgeType.

Definition at line 190 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_values.

16.3.4.26 template<int D, FUNCT_TYPE FuncType> virtual double∗ **teb::BaseEdge**< D, FuncType >::valuesData ( ) [inline], [virtual]

Implements teb::EdgeType.

Definition at line 191 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_values.

16.3.4.27 virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline], [virtual], [inherited]

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References teb::EdgeType::dimension(), and teb::EdgeType::valuesData().

16.3.4.28 template<int D, FUNCT_TYPE FuncType> virtual const std::vector<VertexType*>& teb::BaseEdge< D, FuncType >::vertices ( ) const [inline], [virtual]

Returns

(Read-only) reference to the vertex container.

Definition at line 256 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.3.4.29 template<int D, FUNCT_TYPE FuncType> virtual std::vector<VertexType*>& teb::BaseEdge< D, FuncType >::vertices ( ) [inline], [virtual]

Returns

Reference to the vertex container.

Definition at line 262 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.3.5 Member Data Documentation

16.3.5.1 HessianWorkspace teb::EdgeType::_hessians [protected], [inherited]

Definition at line 178 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), allocateMemory(), teb::EdgeMinimizeTime::computeHessian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeHessian(), computeHessian(), and teb::EdgeType::hessians().

16.3.5.2 BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup [protected], [inherited]

Definition at line 180 of file graph.h.

Referenced by teb::EdgeType::backupJacobian(), teb::EdgeType::discardJacobianBackup(), teb::EdgeType::getJacobianBackup(), and teb::EdgeType::restoreJacobian().

16.3.5.3 JacobianWorkspace teb::EdgeType::_jacobians [protected], [inherited]

Definition at line 173 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), allocateMemory(), teb::EdgeType::backupJacobian(), computeHessian(), teb::EdgeMinimizeTime::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), computeJacobian(), teb::EdgeType::jacobians(), and teb::EdgeType::restoreJacobian().

16.3.5.4 template<int D, FUNCT_TYPE FuncType> ValueVector teb::BaseEdge< D, FuncType >::_values = ValueVector::Zero() [protected]

Definition at line 276 of file base_edge.h.

16.3.5.5 template<int D, FUNCT_TYPE FuncType> std::vector<VertexType*> teb::BaseEdge< D, FuncType >::_vertices [protected]

Definition at line 278 of file base_edge.h.

16.3.5.6 template<int D, FUNCT_TYPE FuncType> std::vector<int> teb::BaseEdge< D, FuncType >::_vertices_dim [protected]

Definition at line 279 of file base_edge.h.

16.3.5.7 template<int D, FUNCT_TYPE FuncType> const int teb::BaseEdge< D, FuncType >::Dimension = D [static]

Definition at line 176 of file base_edge.h.

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

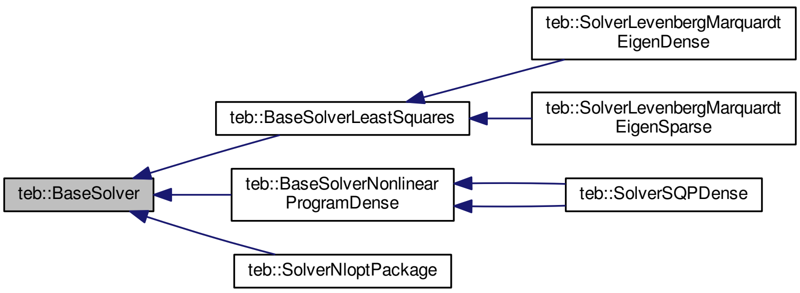
- base_edge.h
- base_edge.hpp

16.4 teb::BaseSolver Class Reference

Base class for solver implementations.

`#include <base_solver.h>`

Inheritance diagram for teb::BaseSolver:



Public Types

- using VertexContainer = HyperGraph::VertexContainer

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).
- using EdgeContainer = HyperGraph::EdgeContainer

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).

Public Member Functions

- BaseSolver ()

  Empty Constructor.
- virtual ∼BaseSolver ()

  Empty Destructor.
- void setConfig (const Config ∗config)

  Set config including solver settings.
- virtual bool solve (HyperGraph ∗optimizable_graph)

  Solve an optimization problem defined by an hyper-graph.

Public Attributes

- const Config ∗ cfg = nullptr

  Store pointer to Config object.
- EIGEN_MAKE_ALIGNED_OPERATOR_NEW

Protected Member Functions

- virtual bool solveImpl ()=0

  Solve the actual optimization problem.
- virtual void initWorkspaces ()

  Initialize workspaces for the block jacobians and hessians.
- void applyIncrement (const Eigen::Ref< const Eigen::VectorXd > &delta)

  Apply a new increment optained by a local optimization to the active vertices.
- void applyOptVec (const Eigen::Ref< const Eigen::VectorXd > &opt_vec)

  Overwrite TEB states and control inputs with new values.
- Eigen::VectorXd getOptVecCopy () const

  Get a copy of the optimization vector (TEB states, control inputs and dt)
- int getOptVecDimension () const

  Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).
- int getDimObjectives () const

  Get dimension of the objective function.
- int getDimEqualities () const

  Get dimension of the equality constraints.
- int getDimInequalities () const

  Get dimension of the inequality constraints.
- void backupVertices ()

  Backup all active vertices.
- void restoreVertices ()

  Restore all values of active vertices from the backup stacks.
- void restoreVerticesButKeepBackup ()

  Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.
- void discardBackupVertices ()

  Discard all values of active vertices from the backup stacks.

Protected Attributes

- VertexContainer ∗ _active_vertices = nullptr

  Pointer to active vertex container (active = non-fixed)
- EdgeContainer ∗ _objectives = nullptr

  Pointer to edges representing objective functions.
- EdgeContainer ∗ _equalities = nullptr

  Pointer to edges representing equality constraints.
- EdgeContainer ∗ _inequalities = nullptr

  Pointer to edges representing inequality constraints.
- int _opt_vec_dim = -1

  Store dimension of the optimziation vector here (see getOptVecDimension()).
- int _objective_dim = -1

  Store dimension of the objective function (see getDimObjectives()).
- int _equalities_dim = -1

  Store dimension of the equality constraints (see getDimEqualities()).
- int _inequalities_dim = -1

  Store dimension of the inequality constraints (see getDimInequalities()).
- unsigned int _no_vert_backups = 0

  Track number of vertices backups made.
- bool _graph_structure_modified = true

  Mark if a new graph structure is available, thus we need to reinitialze all workspaces.

16.4.1 Detailed Description

This abstract class defines the interface for general solvers that are suited for solving the TebController optimization problem.

The solver needs to handle the optimziation problem as a hyper-graph (see EdgeType and VertexType).

Author

Christoph Rösmann (`christoph.roesmann@tu-dortmund.de`)

Examples:

integrator_system_sfun.cpp.

Definition at line 29 of file base_solver.h.

16.4.2 Member Typedef Documentation

16.4.2.1 using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer

Definition at line 36 of file base_solver.h.

16.4.2.2 using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer

Definition at line 34 of file base_solver.h.

16.4.3 Constructor & Destructor Documentation

16.4.3.1 teb::BaseSolver::BaseSolver ( ) `[inline]`

Definition at line 39 of file base_solver.h.

16.4.3.2 virtual teb::BaseSolver::∼BaseSolver ( ) [inline], [virtual]

Definition at line 42 of file base_solver.h.

16.4.4 Member Function Documentation

16.4.4.1 void teb::BaseSolver::applyIncrement (const Eigen::Ref< const Eigen::VectorXd > & *delta*) [inline], [protected]

This method iterates the active vertices container and calls VertexType::plusFree() for each vertex.

Parameters

delta	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.

Definition at line 127 of file base_solver.h.

References _active_vertices.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.4.4.2 void teb::BaseSolver::applyOptVec (const Eigen::Ref< const Eigen::VectorXd > & *opt_vec*) [inline], [protected]

This method iterates the active vertices container and calls VertexType::setFree() for each vertex.

Parameters

opt_vec	Eigen::Vector containing all values. The length equals the dimension of all free variables.

Definition at line 144 of file base_solver.h.

References _active_vertices.

16.4.4.3 void teb::BaseSolver::backupVertices ( ) [inline], [protected]

See Also

VertexType::push()

Definition at line 223 of file base_solver.h.

References _active_vertices, and _no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.4.4.4 void teb::BaseSolver::discardBackupVertices ( ) [inline], [protected]

See Also

VertexType::discardTop()

Definition at line 229 of file base_solver.h.

References _active_vertices, and _no_vert_backups.

Referenced by solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.4.4.5 int teb::BaseSolver::getDimEqualities ( ) const [inline], [protected]

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file base_solver.h.

References _equalities.

Referenced by solve().

16.4.4.6 int teb::BaseSolver::getDimInequalities ( ) const [inline], [protected]

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file base_solver.h.

References _inequalities.

Referenced by solve().

16.4.4.7 int teb::BaseSolver::getDimObjectives ( ) const [inline], [protected]

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file base_solver.h.

References _objectives.

Referenced by solve().

16.4.4.8 Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const [inline], [protected]

This method iterates the active vertices container and calls VertexType::getDataFree() for each vertex.

Remarks

Make sure BaseSolver::_opt_vec_dim is valid (see solve()).

Returns

Eigen::Vector containing all values. The length equals the dimension of all free variables (getOptVecDimension()).

Definition at line 162 of file base_solver.h.

References _active_vertices, and _opt_vec_dim.

16.4.4.9 int teb::BaseSolver::getOptVecDimension ( ) const [inline], [protected]

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

See Also

TebController::getActiveVertices(), getValueDimension()

Definition at line 180 of file base_solver.h.

References _active_vertices.

Referenced by solve().

16.4.4.10 virtual void teb::BaseSolver::initWorkspaces ( ) [inline], [protected], [virtual]

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each EdgeType) or map to existing memory.

Reimplemented in teb::BaseSolverNonlinearProgramDense, teb::SolverNloptPackage, and teb::BaseSolverLeastSquares.

Definition at line 115 of file base_solver.h.

Referenced by solve().

16.4.4.11 void teb::BaseSolver::restoreVertices ( ) [inline], [protected]

See Also

VertexType::pop()

Definition at line 225 of file base_solver.h.

References _active_vertices, and _no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.4.4.12 void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline], [protected]

See Also

VertexType::top()

Definition at line 227 of file base_solver.h.

References _active_vertices.

Referenced by teb::SolverSQPDense::solveImpl().

16.4.4.13 void teb::BaseSolver::setConfig (const Config ∗ *config*) [inline]

Remarks

This function is called from the TEB class within setSolver method.

Parameters

config	Pointer to Config object.

Definition at line 49 of file base_solver.h.

References cfg.

Referenced by teb::TebController$<$ p, q $>$::setSolver().

16.4.4.14 virtual bool teb::BaseSolver::solve (HyperGraph ∗ *optimizable_graph*) [inline], [virtual]

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with solveImpl().

Todo Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

Parameters

optimizable_-graph	pointer to the HyperGraph that should be solved

Definition at line 61 of file base_solver.h.

References _active_vertices, _equalities, _equalities_dim, _graph_structure_modified, _inequalities, _inequalities-_dim, _no_vert_backups, _objective_dim, _objectives, _opt_vec_dim, teb::HyperGraph::activeVertices(), discard-BackupVertices(), teb::HyperGraph::equalities(), getDimEqualities(), getDimInequalities(), getDimObjectives(), get-OptVecDimension(), teb::HyperGraph::inequalities(), initWorkspaces(), teb::HyperGraph::isGraphModified(), teb::-HyperGraph::objectives(), and solveImpl().

16.4.4.15 virtual bool teb::BaseSolver::solveImpl ( ) [protected], [pure virtual]

Store results to the vertices of the active vertices container.

Return values

true	optimization was successfull.
false	optimization was not successfull.

Implemented in teb::SolverSQPDense, teb::SolverNloptPackage, teb::SolverSQPDense, teb::BaseSolverNonlinear-ProgramDense, teb::SolverLevenbergMarquardtEigenSparse, teb::BaseSolverLeastSquares, and teb::Solver-LevenbergMarquardtEigenDense.

Referenced by solve().

16.4.5 Member Data Documentation

16.4.5.1 VertexContainer* teb::BaseSolver::_active_vertices = nullptr [protected]

Definition at line 233 of file base_solver.h.

Referenced by applyIncrement(), applyOptVec(), backupVertices(), discardBackupVertices(), getOptVecCopy(), get-OptVecDimension(), restoreVertices(), restoreVerticesButKeepBackup(), and solve().

16.4.5.2 EdgeContainer* teb::BaseSolver::_equalities = nullptr [protected]

Definition at line 235 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolver-NonlinearProgramDense::buildEqualityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense-::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares-::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::-SolverNloptPackage::constraintsEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), get-DimEqualities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb-::BaseSolverNonlinearProgramDense::initWorkspaces(), and solve().

16.4.5.3 int teb::BaseSolver::_equalities_dim = -1 [protected]

Definition at line 240 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolver-NonlinearProgramDense::buildEqualityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculate-LagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::SolverNloptPackage-::getEqConstrDimFromStorage(), teb::BaseSolverLeastSquares::getValueDimension(), teb::BaseSolverNonlinear-ProgramDense::initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), solve(), and teb::-SolverSQPDense::solveImpl().

16.4.5.4 bool teb::BaseSolver::_graph_structure_modified = true [protected]

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the solve() method.

Definition at line 250 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), solve(), and teb::SolverSQPDense::solveImpl().

16.4.5.5 EdgeContainer* teb::BaseSolver::_inequalities = nullptr [protected]

Definition at line 236 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsInEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), getDimInequalities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and solve().

16.4.5.6 int teb::BaseSolver::_inequalities_dim = -1 [protected]

Definition at line 241 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::BaseSolverLeastSquares::getValueDimension(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), solve(), and teb::SolverSQPDense::solveImpl().

16.4.5.7 unsigned int teb::BaseSolver::_no_vert_backups = 0 [protected]

Definition at line 243 of file base_solver.h.

Referenced by backupVertices(), discardBackupVertices(), restoreVertices(), and solve().

16.4.5.8 int teb::BaseSolver::_objective_dim = -1 [protected]

Definition at line 239 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::getValueDimension(), and solve().

16.4.5.9 EdgeContainer* teb::BaseSolver::_objectives = nullptr [protected]

Definition at line 234 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::buildObjectiveValue(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), getDimObjectives(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), teb::SolverNloptPackage::objectives(), and solve().

16.4.5.10 int teb::BaseSolver::_opt_vec_dim = -1 [protected]

Definition at line 238 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), getOptVecCopy(), teb::SolverNloptPackage::getOptVecDimFromStorage(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.4.5.11 const Config∗ teb::BaseSolver::cfg = nullptr

See Also

setConfig()

Definition at line 95 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), setConfig(), and teb::SolverSQPDense::solveImpl().

16.4.5.12 teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW

Definition at line 253 of file base_solver.h.

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

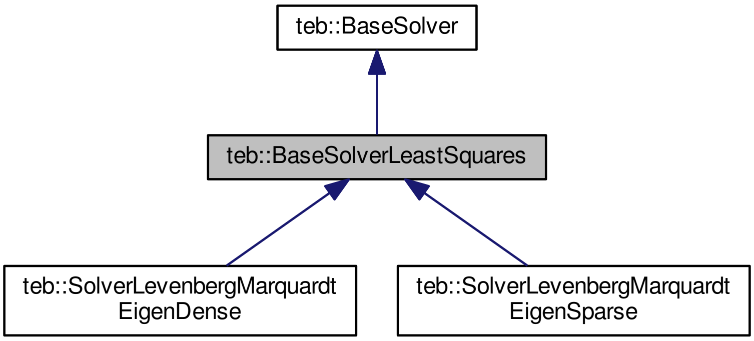
- base_solver.h

16.5 teb::BaseSolverLeastSquares Class Reference

Extended base solver class for least squares optimizations.

`#include <base_solver_least_squares.h>`

Inheritance diagram for teb::BaseSolverLeastSquares:



Public Types

- using VertexContainer = HyperGraph::VertexContainer

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).
- using EdgeContainer = HyperGraph::EdgeContainer

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).

Public Member Functions

- BaseSolverLeastSquares ()

  Empty Constructor.
- void setConfig (const Config ∗config)

  Set config including solver settings.
- virtual bool solve (HyperGraph ∗optimizable_graph)

  Solve an optimization problem defined by an hyper-graph.

Public Attributes

- const Config ∗ cfg = nullptr

  Store pointer to Config object.
- EIGEN_MAKE_ALIGNED_OPERATOR_NEW

Protected Member Functions

- virtual bool solveImpl ()=0

  Solve the actual optimization problem.
- virtual void initWorkspaces ()

  Initialize workspaces for the block jacobians and hessians.
- void buildValueVector ()

  Build value / cost vector including all objectives and constraints.
- int getValueDimension () const

  Get dimension of the composed value/cost vector (including objectives and constraints).

- double getChi2 () const

  Calculate the χ^2 error of the optimization problem.
- void adaptWeights ()

  Automatic weight adaptation that increases soft constraint weights after each outer teb iteration.
- void applyIncrement (const Eigen::Ref< const Eigen::VectorXd > &delta)

  Apply a new increment optained by a local optimization to the active vertices.
- void applyOptVec (const Eigen::Ref< const Eigen::VectorXd > &opt_vec)

  Overwrite TEB states and control inputs with new values.
- Eigen::VectorXd getOptVecCopy () const

  Get a copy of the optimization vector (TEB states, control inputs and dt)
- int getOptVecDimension () const

  Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).
- int getDimObjectives () const

  Get dimension of the objective function.
- int getDimEqualities () const

  Get dimension of the equality constraints.
- int getDimInequalities () const

  Get dimension of the inequality constraints.
- void backupVertices ()

  Backup all active vertices.
- void restoreVertices ()

  Restore all values of active vertices from the backup stacks.
- void restoreVerticesButKeepBackup ()

  Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.
- void discardBackupVertices ()

  Discard all values of active vertices from the backup stacks.

Protected Attributes

- Eigen::VectorXd _values

  Store the value vector computed in buildValueVector().
- int _weight_adapt_count = 0

  Store current state of the weight adaptation method adaptWeights().
- double _weight_equalities = 1

  Store current weight for equality soft-constraints.
- double _weight_inequalities = 1

  Store current weight for inequality soft-constraints.
- int _val_dim = -1

  Store dimension of the value vector here (see getValueDimension()).
- VertexContainer ∗ _active_vertices = nullptr

  Pointer to active vertex container (active = non-fixed)
- EdgeContainer ∗ _objectives = nullptr

  Pointer to edges representing objective functions.
- EdgeContainer ∗ _equalities = nullptr

  Pointer to edges representing equality constraints.
- EdgeContainer ∗ _inequalities = nullptr

  Pointer to edges representing inequality constraints.
- int _opt_vec_dim = -1

  Store dimension of the optimziation vector here (see getOptVecDimension()).
- int _objective_dim = -1

Store dimension of the objective function (see getDimObjectives()).

- int _equalities_dim = -1

  Store dimension of the equality constraints (see getDimEqualities()).
- int _inequalities_dim = -1

  Store dimension of the inequality constraints (see getDimInequalities()).
- unsigned int _no_vert_backups = 0

  Track number of vertices backups made.
- bool _graph_structure_modified = true

  Mark if a new graph structure is available, thus we need to reinitialze all workspaces.

16.5.1 Detailed Description

This abstract class extends the BaseSolver class by routines and methods required by dedicated least squares solvers.

It transforms inequality and equality constraints into soft constraints that are added to the objective functions. objective functions are squared.

The jacobian calculation is based on the new transformed objective function and used for quasi-newton hessian approximation $\mathbf{H} = \mathbf{J}^T \mathbf{J}$. However, the actual hessian calculation using the computeHessian() functions of each edge is ommitted.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Definition at line 29 of file base_solver_least_squares.h.

16.5.2 Member Typedef Documentation

16.5.2.1 using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer [inherited]

Definition at line 36 of file base_solver.h.

16.5.2.2 using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer [inherited]

Definition at line 34 of file base_solver.h.

16.5.3 Constructor & Destructor Documentation

16.5.3.1 teb::BaseSolverLeastSquares::BaseSolverLeastSquares ( ) [inline]

Definition at line 34 of file base_solver_least_squares.h.

16.5.4 Member Function Documentation

16.5.4.1 void teb::BaseSolverLeastSquares::adaptWeights ( ) [protected]

This method increases the soft constraint weights for inequalities and equalities after each outer TEB optimization loop (see TebController::optimizeTEB()).

After the outer TEB loop is completed (Config::Teb::teb_iter), the weights are set back to Config::Optim::Solver::Lsq::weight_equalities and Config::Optim::Solver::Lsq::weight_inequalities.

The weights are increased by $\sigma = \sigma_{init} \cdot \gamma^i$. γ denotes the increasement factor Config::Optim::Solver::Lsq::weight_-adaptation_factor. i denotes the current iteration number. σ_{init} is obtained from the config (see above).

Call this method at the beginning of solveImpl()!

Remarks

This procedere forces the optimization result to first contract the trajectory in sense of the objective, and afterwards trying to satisfy the constraints. Starting with very high weights in advance could lead to abrupt gradients for the solver. In addition the effect for the objectives (e.g. time optimallity) could be slow, if the corresponding gradients are extremly small in comparision to constraint gradients.

See Also

TebController::optimizeTEB()

Definition at line 100 of file base_solver_least_squares.cpp.

References _weight_adapt_count, _weight_equalities, _weight_inequalities, teb::BaseSolver::cfg, teb::Config::-Optim::Solver::lsq, teb::Config::optim, teb::Config::Optim::solver, teb::Config::teb, teb::Config::Teb::teb_iter, teb::-Config::Optim::Solver::Lsq::weight_adaptation_factor, teb::Config::Optim::Solver::Lsq::weight_equalities, and teb::-Config::Optim::Solver::Lsq::weight_inequalities.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::SolverLevenbergMarquardt-EigenSparse::solveImpl().

16.5.4.2 void teb::BaseSolver::applyIncrement (const Eigen::Ref< const Eigen::VectorXd > & *delta*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::plusFree() for each vertex.

Parameters

delta	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.

Definition at line 127 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigen-Sparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.5.4.3 void teb::BaseSolver::applyOptVec (const Eigen::Ref< const Eigen::VectorXd > & *opt_vec*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::setFree() for each vertex.

Parameters

opt_vec	Eigen::Vector containing all values. The length equals the dimension of all free variables.

Definition at line 144 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

16.5.4.4 void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::push()

Definition at line 223 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.5.4.5 void teb::BaseSolverLeastSquares::buildValueVector ( ) [protected]

This method constructs the full value/cost vector $\mathbf{f}$ (not $\mathbf{f}^2$) for the underlying least-squares optimization problem.

The lenght of the vector can be obtained using getValueDimension().

Constraints are transformed into costs/objectives using soft constraints:

- Equality constraints $c(x) = 0$.

  These constraints are directly taken as objectives since $c^2(x)$ has a local minima at $x = 0$.

- Inequality constraints $c(x) \leq 0$

  These constraints are transformed using $f = \max\left(c(x), 0\right)$ (component-wise).

  This notation is similar to: c(x) <= epsilon ? 0 : c(x).

  Afterwards (in the actual solveImpl() method, the cost function will be squared, that leads to twice differnetiable costs for the constraints, if f is piecewise differentiable once and if it intersectects with the abscissa.

In addition the weights BaseSolverLeastSquares::_weight_equalities and BaseSolverLeastSquares::_weight_inequalities are taken into account in order to weight the "soft constraint objectives". To make the weights comparable to [2] and our Matlab TEB version, we take the square root of both weights, since the least-square problem here is formulated as $f^2(x, \sigma)$ instead of $\sigma f^2(x)$. σ denotes the weight.

The results are stored internally to BaseSolverLeastSquares::_values.

See Also

adaptWeights(), getValueDimension()

Definition at line 31 of file base_solver_least_squares.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_inequalities, teb::BaseSolver::_objectives, _val_dim, _values, _weight_equalities, _weight_inequalities, teb::BaseSolver::cfg, teb::Config::Optim::Solver::lsq, teb::Config::optim, teb::Config::Optim::Solver::Lsq::soft_constr_epsilon, and teb::Config::Optim::solver.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.5.4.6 void teb::BaseSolver::discardBackupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::discardTop()

Definition at line 229 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.5.4.7 double teb::BaseSolverLeastSquares::getChi2 ( ) const [inline], [protected]

The χ^2 error is the scalar cost value of the least-squares optimization problem since the total cost function is composed of weighted terms:

$$f(\mathscr{B}) = \mathbf{f}(\mathscr{B})^T \mathbf{f}(\mathscr{B})$$

Weights are included in $\mathbf{f}(\mathscr{B})$. In this case it is $\chi^2 = f(\mathscr{B})$.

Todo Maybe switch to the formulation $f(\mathscr{B}) = \mathbf{f}(\mathscr{B})^T \Omega \mathbf{f}(\mathscr{B})$ similar to g2o and Teb-Matlab.

Definition at line 82 of file base_solver_least_squares.h.

References _values.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::SolverLevenbergMarquardt-EigenSparse::solveImpl().

16.5.4.8 int teb::BaseSolver::getDimEqualities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file base_solver.h.

References teb::BaseSolver::_equalities.

Referenced by teb::BaseSolver::solve().

16.5.4.9 int teb::BaseSolver::getDimInequalities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file base_solver.h.

References teb::BaseSolver::_inequalities.

Referenced by teb::BaseSolver::solve().

16.5.4.10 int teb::BaseSolver::getDimObjectives ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file base_solver.h.

References teb::BaseSolver::_objectives.

Referenced by teb::BaseSolver::solve().

16.5.4.11 Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::getDataFree() for each vertex.

Remarks

Make sure BaseSolver::_opt_vec_dim is valid (see solve()).

Returns

Eigen::Vector containing all values. The length equals the dimension of all free variables (getOptVec-Dimension()).

Definition at line 162 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_opt_vec_dim.

16.5.4.12 int teb::BaseSolver::getOptVecDimension ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

See Also

TebController::getActiveVertices(), getValueDimension()

Definition at line 180 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::BaseSolver::solve().

16.5.4.13 int teb::BaseSolverLeastSquares::getValueDimension ( ) const [protected]

This method returns the dimension of the full cost/value vector including all objective and constraints.

In particular it sums up all dimensions of single edges stored in the given hyper-graph.

Remarks

Make sure BaseSolver::_objective_dim, BaseSolver::_equalities_dim and BaseSolver::_inequalities_dim are valid (see solve()).

Returns

dimension of the complete value/cost vector.

See Also

buildValueVector(), getOptVecDimension(), getDimObjectives(), getDimEqualities(), getDimInequalities()

Definition at line 73 of file base_solver_least_squares.cpp.

References teb::BaseSolver::_equalities_dim, teb::BaseSolver::_inequalities_dim, and teb::BaseSolver::_objective_dim.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.5.4.14 virtual void teb::BaseSolverLeastSquares::initWorkspaces ( ) [inline], [protected], [virtual]

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each EdgeType) or map to existing memory.

Reimplemented from teb::BaseSolver.

Definition at line 43 of file base_solver_least_squares.h.

References teb::BaseSolver::_equalities, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_objectives, PRINT_DEBUG_COND_ONCE, and teb::QUADRATIC.

16.5.4.15 void teb::BaseSolver::restoreVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::pop()

Definition at line 225 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.5.4.16 void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline], [protected], [inherited]

See Also

VertexType::top()

Definition at line 227 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverSQPDense::solveImpl().

16.5.4.17 void teb::BaseSolver::setConfig (const Config * *config*) [inline], [inherited]

Remarks

This function is called from the TEB class within setSolver method.

Parameters

config	Pointer to Config object.

Definition at line 49 of file base_solver.h.

References teb::BaseSolver::cfg.

Referenced by teb::TebController< p, q >::setSolver().

16.5.4.18 virtual bool teb::BaseSolver::solve (HyperGraph * *optimizable_graph*) [inline], [virtual], [inherited]

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with solveImpl().

Todo Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

Parameters

optimizable_graph	pointer to the HyperGraph that should be solved

Definition at line 61 of file base_solver.h.

References teb::BaseSolver::_active_vertices, teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, teb::BaseSolver::_no_vert_backups, teb::BaseSolver::_objective_dim, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::HyperGraph::activeVertices(), teb::BaseSolver::discardBackupVertices(), teb::HyperGraph::equalities(), teb::BaseSolver::getDimEqualities(), teb::BaseSolver::getDimInequalities(), teb::BaseSolver::getDimObjectives(), teb::BaseSolver::getOptVecDimension(), teb::HyperGraph::inequalities(), teb::BaseSolver::initWorkspaces(), teb::HyperGraph::isGraphModified(), teb::HyperGraph::objectives(), and teb::BaseSolver::solveImpl().

16.5.4.19 virtual bool teb::BaseSolverLeastSquares::solveImpl ( ) [protected], [pure virtual]

Store results to the vertices of the active vertices container.

Return values

true	optimization was successfull.
false	optimization was not successfull.

Implements teb::BaseSolver.

Implemented in teb::SolverLevenbergMarquardtEigenSparse, and teb::SolverLevenbergMarquardtEigenDense.

16.5.5 Member Data Documentation

16.5.5.1 VertexContainer∗ teb::BaseSolver::_active_vertices = nullptr [protected], [inherited]

Definition at line 233 of file base_solver.h.

Referenced by teb::BaseSolver::applyIncrement(), teb::BaseSolver::applyOptVec(), teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::getOptVecCopy(), teb::BaseSolver::getOptVecDimension(), teb::BaseSolver::restoreVertices(), teb::BaseSolver::restoreVerticesButKeepBackup(), and teb::BaseSolver::solve().

16.5.5.2 EdgeContainer∗ teb::BaseSolver::_equalities = nullptr [protected], [inherited]

Definition at line 235 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimEqualities(), initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and teb::BaseSolver::solve().

16.5.5.3 int teb::BaseSolver::_equalities_dim = -1 [protected], [inherited]

Definition at line 240 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::SolverNloptPackage::getEqConstrDimFromStorage(), getValueDimension(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.5.5.4 bool teb::BaseSolver::_graph_structure_modified = true [protected], [inherited]

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the solve() method.

Definition at line 250 of file base_solver.h.

Referenced by initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.5.5.5 EdgeContainer∗ teb::BaseSolver::_inequalities = nullptr [protected], [inherited]

Definition at line 236 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsInEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimInequalities(), initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and teb::BaseSolver::solve().

16.5.5.6 int teb::BaseSolver::_inequalities_dim = -1 [protected], [inherited]

Definition at line 241 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), getValueDimension(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.5.5.7 unsigned int teb::BaseSolver::_no_vert_backups = 0 [protected], [inherited]

Definition at line 243 of file base_solver.h.

Referenced by teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::restoreVertices(), and teb::BaseSolver::solve().

16.5.5.8 int teb::BaseSolver::_objective_dim = -1 [protected], [inherited]

Definition at line 239 of file base_solver.h.

Referenced by getValueDimension(), and teb::BaseSolver::solve().

16.5.5.9 EdgeContainer* teb::BaseSolver::_objectives = nullptr [protected], [inherited]

Definition at line 234 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::buildObjectiveValue(), buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimObjectives(), initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::SolverNloptPackage::objectives(), and teb::BaseSolver::solve().

16.5.5.10 int teb::BaseSolver::_opt_vec_dim = -1 [protected], [inherited]

Definition at line 238 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getOptVecCopy(), teb::SolverNloptPackage::getOptVecDimFromStorage(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.5.5.11 int teb::BaseSolverLeastSquares::_val_dim = -1 [protected]

Definition at line 96 of file base_solver_least_squares.h.

Referenced by teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian(), teb::SolverLevenberg-MarquardtEigenDense::buildJacobian(), buildValueVector(), teb::SolverLevenbergMarquardtEigenDense::solve-Impl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.5.5.12 Eigen::VectorXd teb::BaseSolverLeastSquares::_values [protected]

Definition at line 90 of file base_solver_least_squares.h.

Referenced by buildValueVector(), getChi2(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::-SolverLevenbergMarquardtEigenSparse::solveImpl().

16.5.5.13 int teb::BaseSolverLeastSquares::_weight_adapt_count = 0 [protected]

Definition at line 92 of file base_solver_least_squares.h.

Referenced by adaptWeights().

16.5.5.14 double teb::BaseSolverLeastSquares::_weight_equalities = 1 [protected]

Definition at line 93 of file base_solver_least_squares.h.

Referenced by adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::Solver-LevenbergMarquardtEigenSparse::buildJacobian(), and buildValueVector().

16.5.5.15 double teb::BaseSolverLeastSquares::_weight_inequalities = 1 [protected]

Definition at line 94 of file base_solver_least_squares.h.

Referenced by adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::Solver-LevenbergMarquardtEigenSparse::buildJacobian(), and buildValueVector().

16.5.5.16 const Config∗ teb::BaseSolver::cfg = nullptr [inherited]

See Also

setConfig()

Definition at line 95 of file base_solver.h.

Referenced by adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::Solver-LevenbergMarquardtEigenSparse::buildJacobian(), buildValueVector(), teb::BaseSolverNonlinearProgramDense-::calculateLagrangianHessian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFG-S(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::setConfig(), and teb::SolverSQPDense::solveImpl().

16.5.5.17 teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW [inherited]

Definition at line 253 of file base_solver.h.

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

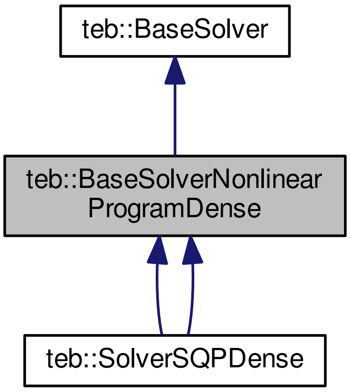
- base_solver_least_squares.h
- base_solver_least_squares.cpp

16.6 teb::BaseSolverNonlinearProgramDense Class Reference

Extended base solver class for nonlinear programs (dense version).

`#include <base_solver_nonlinear_program_dense.h>`

Inheritance diagram for teb::BaseSolverNonlinearProgramDense:



Public Types

- using VertexContainer = HyperGraph::VertexContainer

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).
- using EdgeContainer = HyperGraph::EdgeContainer

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).

Public Member Functions

- BaseSolverNonlinearProgramDense ()

  Empty Constructor.
- void setConfig (const Config ∗config)

  Set config including solver settings.
- virtual bool solve (HyperGraph ∗optimizable_graph)

  Solve an optimization problem defined by an hyper-graph.

Public Attributes

- const Config ∗ cfg = nullptr

  Store pointer to Config object.
- EIGEN_MAKE_ALIGNED_OPERATOR_NEW

Protected Types

- using MatMapRowMajor = Eigen::Map< Eigen::Matrix< double,-1,-1, Eigen::RowMajor >>

Protected Member Functions

- virtual bool solveImpl ()=0

    Solve the actual optimization problem.
- virtual void initWorkspaces ()

    Initialize workspaces for the block jacobians and hessians.
- virtual void initSolverWorkspace ()=0
- void initializeLagrangeMultiplier ()
- void buildObjectiveValue ()

    Get the sum of all objective values since we solve $f = \min \sum f_k$.
- void buildEqualityConstraintValueVector ()

    Build value / cost vector including all equality constraints.
- void buildInequalityConstraintValueVector ()

    Build value / cost vector including all inequality constraints.
- void buildObjectiveGradient ()
- void buildEqualityConstraintJacobian ()
- void buildInequalityConstraintJacobian ()
- void calculateLagrangianGradient ()
- void calculateLagrangianHessian (Eigen::VectorXd $*$increment=nullptr)
- void calculateLagrangianHessianNumerically ()
- void initHessianBFGS ()
- void calculateLagrangianHessianFullBFGS (Eigen::VectorXd $*$increment)
- void applyIncrement (const Eigen::Ref$<$ const Eigen::VectorXd $>$ &delta)

    Apply a new increment optained by a local optimization to the active vertices.
- void applyOptVec (const Eigen::Ref$<$ const Eigen::VectorXd $>$ &opt_vec)

    Overwrite TEB states and control inputs with new values.
- Eigen::VectorXd getOptVecCopy () const

    Get a copy of the optimization vector (TEB states, control inputs and dt)
- int getOptVecDimension () const

    Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).
- int getDimObjectives () const

    Get dimension of the objective function.
- int getDimEqualities () const

    Get dimension of the equality constraints.
- int getDimInequalities () const

    Get dimension of the inequality constraints.
- void backupVertices ()

    Backup all active vertices.
- void restoreVertices ()

    Restore all values of active vertices from the backup stacks.
- void restoreVerticesButKeepBackup ()

    Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.
- void discardBackupVertices ()

    Discard all values of active vertices from the backup stacks.

Protected Attributes

- double _objective_value

  Store the sum of all values computed in buildObjectiveValueVector() since we solve $f = \min \sum f_k$.
- Eigen::VectorXd _equality_values

  Store the value vector computed in buildEqualityConstraintValueVector().
- Eigen::VectorXd _inequality_values

  Store the value vector computed in buildInequalityConstraintValueVector().
- Eigen::VectorXd _objective_gradient

  Store the gradient of the objective value computed in buildObjectiveGradient().
- MatMapRowMajor _equality_jacobian = MatMapRowMajor(nullptr,0,0)

  Store the jacobian matrix of the equality constraint computed in buildEqualityConstraintJacobian().
- MatMapRowMajor _inequality_jacobian = MatMapRowMajor(nullptr,0,0)

  Store the jacobian matrix of the inequality constraint computed in buildInequalityConstraintJacobian().
- Eigen::VectorXd _lagrangian_gradient
- Eigen::VectorXd _lagrangian_gradient_backup

  The gradient from the last step has to be stored for BFGS.
- Eigen::Matrix$<$ double,-1,-1,
  Eigen::RowMajor $>$ _lagrangian_hessian

  Store the hessian of the lagrangian $\nabla^2 L = \nabla^2 (f - \mu^T \mathbf{ceq} - \lambda^T \mathbf{c})$.
- Eigen::VectorXd _multiplier_ineq
- Eigen::VectorXd _multiplier_eq
- VertexContainer $*$ _active_vertices = nullptr

  Pointer to active vertex container (active = non-fixed)
- EdgeContainer $*$ _objectives = nullptr

  Pointer to edges representing objective functions.
- EdgeContainer $*$ _equalities = nullptr

  Pointer to edges representing equality constraints.
- EdgeContainer $*$ _inequalities = nullptr

  Pointer to edges representing inequality constraints.
- int _opt_vec_dim = -1

  Store dimension of the optimziation vector here (see getOptVecDimension()).
- int _objective_dim = -1

  Store dimension of the objective function (see getDimObjectives()).
- int _equalities_dim = -1

  Store dimension of the equality constraints (see getDimEqualities()).
- int _inequalities_dim = -1

  Store dimension of the inequality constraints (see getDimInequalities()).
- unsigned int _no_vert_backups = 0

  Track number of vertices backups made.
- bool _graph_structure_modified = true

  Mark if a new graph structure is available, thus we need to reinitialze all workspaces.

16.6.1 Detailed Description

This abstract class extends the BaseSolver class by routines and methods required by dedicated nonlinear program solvers.

Inequality and equality constraints are treaten as hard constraints. This class provides methods to calculate objective and constraint gradients/jacobians and as well as the Hessian of the Lagrangian.

The actual solveImpl() implementation has to be done in a seperate subclass, e.g. depending if the underlying nonlinear-program solver is a newton-type solver, interior-point, barrier, sqp, ... solver.

Todo The documentation of this class, after this class passed a couple of more tests

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Definition at line 32 of file base_solver_nonlinear_program_dense.h.

16.6.2 Member Typedef Documentation

16.6.2.1 using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer [inherited]

Definition at line 36 of file base_solver.h.

16.6.2.2 using teb::BaseSolverNonlinearProgramDense::MatMapRowMajor = Eigen::Map<Eigen::Matrix<double,-1,-1,Eigen::RowMajor>> [protected]

Definition at line 110 of file base_solver_nonlinear_program_dense.h.

16.6.2.3 using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer [inherited]

Definition at line 34 of file base_solver.h.

16.6.3 Constructor & Destructor Documentation

16.6.3.1 teb::BaseSolverNonlinearProgramDense::BaseSolverNonlinearProgramDense ( ) [inline]

Definition at line 37 of file base_solver_nonlinear_program_dense.h.

16.6.4 Member Function Documentation

16.6.4.1 void teb::BaseSolver::applyIncrement (const Eigen::Ref< const Eigen::VectorXd > & *delta*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::plusFree() for each vertex.

Parameters

delta	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.

Definition at line 127 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.6.4.2 void teb::BaseSolver::applyOptVec (const Eigen::Ref< const Eigen::VectorXd > & *opt_vec*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::setFree() for each vertex.

Parameters

opt_vec	Eigen::Vector containing all values. The length equals the dimension of all free variables.

Definition at line 144 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

16.6.4.3 void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::push()

Definition at line 223 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigen-Sparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.6.4.4 void teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian ( ) [protected]

Definition at line 154 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, _equality_jacobian, _equality_values, teb::BaseSolver::_opt_vec_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::is-FixedAny(), and teb::VertexType::isFixedComp().

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.5 void teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector ( ) [protected]

Definition at line 30 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, and _equality_values.

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.6 void teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian ( ) [protected]

Definition at line 200 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, _inequality_jacobian, _inequality_-values, teb::BaseSolver::_opt_vec_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::Vertex-Type::isFixedAny(), and teb::VertexType::isFixedComp().

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.7 void teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector ( ) [protected]

Definition at line 54 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, and _inequality_values.

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.8 void teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient ( ) [protected]

Definition at line 82 of file base_solver_nonlinear_program_dense.cpp.

References _objective_gradient, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::VertexType-::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), teb::VertexType::isFixedComp(), teb-::LINEAR_SQUARED, and teb::NONLINEAR_SQUARED.

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.9 void teb::BaseSolverNonlinearProgramDense::buildObjectiveValue ( ) [protected]

Definition at line 7 of file base_solver_nonlinear_program_dense.cpp.

References _objective_value, teb::BaseSolver::_objectives, teb::LINEAR_SQUARED, and teb::NONLINEAR_SQUARED.

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.10 void teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient ( ) [inline], [protected]

Definition at line 98 of file base_solver_nonlinear_program_dense.h.

References _equality_jacobian, _inequality_jacobian, _lagrangian_gradient, _multiplier_eq, _multiplier_ineq, and _objective_gradient.

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.11 void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian (Eigen::VectorXd ∗ *increment =* nullptr) [protected]

Definition at line 245 of file base_solver_nonlinear_program_dense.cpp.

References teb::BLOCK_BFGS, calculateLagrangianHessianFullBFGS(), calculateLagrangianHessianNumerically(), teb::BaseSolver::cfg, teb::FULL_BFGS, teb::FULL_BFGS_WITH_STRUCTURE_FILTER, teb::Config::Optim-::Solver::NonlinearProgram::hessian, teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_method, teb::Config::Optim::Solver::nonlin_prog, teb::NUMERIC, teb::Config::optim, PRINT_ERROR, PRINT_INFO, teb::-Config::Optim::solver, and teb::ZERO_HESSIAN.

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.12 void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS (Eigen::VectorXd ∗ *increment*) [protected]

Definition at line 520 of file base_solver_nonlinear_program_dense.cpp.

References _lagrangian_gradient, _lagrangian_gradient_backup, _lagrangian_hessian, teb::Config::Optim::Solver-::NonlinearProgram::Hessian::bfgs_damped_mode, teb::BaseSolver::cfg, teb::Config::Optim::Solver::Nonlinear-Program::hessian, teb::Config::Optim::Solver::nonlin_prog, teb::Config::optim, PRINT_DEBUG, and teb::Config::-Optim::solver.

Referenced by calculateLagrangianHessian().

16.6.4.13 void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically ( ) [protected]

Definition at line 273 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, _equality_values, teb::BaseSolver-::_inequalities, teb::BaseSolver::_inequalities_dim, _inequality_values, _lagrangian_hessian, _multiplier_eq, _-multiplier_ineq, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::VertexType::dimension(), teb::-VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), teb::VertexType::isFixedComp(), teb::LINEAR, teb::LINEAR_SQUARED, teb::NONLINEAR_ONCE_DIFF, and teb::NONLINEAR_SQUARED.

Referenced by calculateLagrangianHessian(), and initHessianBFGS().

16.6.4.14 void teb::BaseSolver::discardBackupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::discardTop()

Definition at line 229 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.6.4.15 int teb::BaseSolver::getDimEqualities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file base_solver.h.

References teb::BaseSolver::_equalities.

Referenced by teb::BaseSolver::solve().

16.6.4.16 int teb::BaseSolver::getDimInequalities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file base_solver.h.

References teb::BaseSolver::_inequalities.

Referenced by teb::BaseSolver::solve().

16.6.4.17 int teb::BaseSolver::getDimObjectives ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file base_solver.h.

References teb::BaseSolver::_objectives.

Referenced by teb::BaseSolver::solve().

16.6.4.18 Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::getDataFree() for each vertex.

Remarks

Make sure BaseSolver::_opt_vec_dim is valid (see solve()).

Returns

Eigen::Vector containing all values. The length equals the dimension of all free variables (getOptVecDimension()).

Definition at line 162 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_opt_vec_dim.

16.6.4.19 int teb::BaseSolver::getOptVecDimension ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

See Also

TebController::getActiveVertices(), getValueDimension()

Definition at line 180 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::BaseSolver::solve().

16.6.4.20 void teb::BaseSolverNonlinearProgramDense::initHessianBFGS ( ) [protected]

Definition at line 489 of file base_solver_nonlinear_program_dense.cpp.

References _lagrangian_gradient, _lagrangian_gradient_backup, _lagrangian_hessian, teb::BaseSolver::_opt_-vec_dim, calculateLagrangianHessianNumerically(), teb::BaseSolver::cfg, teb::Config::Optim::Solver::Nonlinear-Program::hessian, teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init, teb::Config::Optim::-Solver::NonlinearProgram::Hessian::hessian_init_identity_scale, teb::IDENTITY, teb::Config::Optim::Solver::nonlin-_prog, teb::NUMERIC, teb::Config::optim, PRINT_ERROR, teb::Config::Optim::solver, and teb::ZERO.

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.21 void teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier ( ) [inline], [protected]

Definition at line 70 of file base_solver_nonlinear_program_dense.h.

References teb::BaseSolver::_equalities_dim, teb::BaseSolver::_inequalities_dim, _multiplier_eq, and _multiplier_-ineq.

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.22 virtual void teb::BaseSolverNonlinearProgramDense::initSolverWorkspace ( ) [protected], [pure virtual]

Implemented in teb::SolverSQPDense, and teb::SolverSQPDense.

16.6.4.23 virtual void teb::BaseSolverNonlinearProgramDense::initWorkspaces ( ) [inline], [protected], [virtual]

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each Edge-Type) or map to existing memory.

Reimplemented from teb::BaseSolver.

Definition at line 46 of file base_solver_nonlinear_program_dense.h.

References teb::BaseSolver::_equalities, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_-inequalities, teb::BaseSolver::_objectives, teb::LINEAR_SQUARED, teb::NONLINEAR_SQUARED, and PRINT-_DEBUG_COND_ONCE.

16.6.4.24 void teb::BaseSolver::restoreVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::pop()

Definition at line 225 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::SolverLevenbergMarquardt-EigenSparse::solveImpl().

16.6.4.25 void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline], [protected], [inherited]

See Also

VertexType::top()

Definition at line 227 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverSQPDense::solveImpl().

16.6.4.26 void teb::BaseSolver::setConfig (const Config * *config*) [inline], [inherited]

Remarks

This function is called from the TEB class within setSolver method.

Parameters

config	Pointer to Config object.

Definition at line 49 of file base_solver.h.

References teb::BaseSolver::cfg.

Referenced by teb::TebController< p, q >::setSolver().

16.6.4.27 virtual bool teb::BaseSolver::solve (HyperGraph * *optimizable_graph*) [inline], [virtual], [inherited]

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with solveImpl().

Todo Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

Parameters

optimizable_graph	pointer to the HyperGraph that should be solved

Definition at line 61 of file base_solver.h.

References teb::BaseSolver::_active_vertices, teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, teb::BaseSolver::_no_vert_backups, teb::BaseSolver::_objective_dim, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::HyperGraph::activeVertices(), teb::BaseSolver::discardBackupVertices(), teb::HyperGraph::equalities(), teb::BaseSolver::getDimEqualities(), teb::BaseSolver::getDimInequalities(), teb::BaseSolver::getDimObjectives(), teb::BaseSolver::getOptVecDimension(), teb::HyperGraph::inequalities(), teb::BaseSolver::initWorkspaces(), teb::HyperGraph::isGraphModified(), teb::HyperGraph::objectives(), and teb::BaseSolver::solveImpl().

16.6.4.28 virtual bool teb::BaseSolverNonlinearProgramDense::solveImpl ( ) [protected], [pure virtual]

Store results to the vertices of the active vertices container.

Return values

true	optimization was successfull.
false	optimization was not successfull.

Implements teb::BaseSolver.

Implemented in teb::SolverSQPDense, and teb::SolverSQPDense.

16.6.5 Member Data Documentation

16.6.5.1 VertexContainer* teb::BaseSolver::_active_vertices = nullptr [protected], [inherited]

Definition at line 233 of file base_solver.h.

Referenced by teb::BaseSolver::applyIncrement(), teb::BaseSolver::applyOptVec(), teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::getOptVecCopy(), teb::BaseSolver::getOptVecDimension(), teb::BaseSolver::restoreVertices(), teb::BaseSolver::restoreVerticesButKeepBackup(), and teb::BaseSolver::solve().

16.6.5.2 EdgeContainer* teb::BaseSolver::_equalities = nullptr [protected], [inherited]

Definition at line 235 of file base_solver.h.

Referenced by buildEqualityConstraintJacobian(), buildEqualityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimEqualities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), initWorkspaces(), and teb::BaseSolver::solve().

16.6.5.3 int teb::BaseSolver::_equalities_dim = -1 [protected], [inherited]

Definition at line 240 of file base_solver.h.

Referenced by buildEqualityConstraintJacobian(), buildEqualityConstraintValueVector(), calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::SolverNloptPackage::getEqConstrDimFromStorage(), teb::BaseSolverLeastSquares::getValueDimension(), initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.6.5.4 MatMapRowMajor teb::BaseSolverNonlinearProgramDense::_equality_jacobian = MatMapRowMajor(nullptr,0,0) [protected]

Definition at line 117 of file base_solver_nonlinear_program_dense.h.

Referenced by buildEqualityConstraintJacobian(), calculateLagrangianGradient(), teb::SolverSQPDense::calculateMeritDerivative(), and teb::SolverSQPDense::initSolverWorkspace().

16.6.5.5 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_equality_values [protected]

Definition at line 113 of file base_solver_nonlinear_program_dense.h.

Referenced by buildEqualityConstraintJacobian(), buildEqualityConstraintValueVector(), calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMerit(), teb::SolverSQPDense::calculateMeritDerivative(), teb::SolverSQPDense::checkConvergence(), teb::SolverSQPDense::initSolverWorkspace(), and teb::SolverSQPDense::solveImpl().

16.6.5.6 bool teb::BaseSolver::_graph_structure_modified = true [protected], [inherited]

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the solve() method.

Definition at line 250 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::initWorkspaces(), initWorkspaces(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.6.5.7 EdgeContainer* teb::BaseSolver::_inequalities = nullptr [protected], [inherited]

Definition at line 236 of file base_solver.h.

Referenced by buildInequalityConstraintJacobian(), buildInequalityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsInEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimInequalities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), initWorkspaces(), and teb::BaseSolver::solve().

16.6.5.8 int teb::BaseSolver::_inequalities_dim = -1 [protected], [inherited]

Definition at line 241 of file base_solver.h.

Referenced by buildInequalityConstraintJacobian(), buildInequalityConstraintValueVector(), calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::BaseSolverLeastSquares::getValueDimension(), initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.6.5.9 MatMapRowMajor teb::BaseSolverNonlinearProgramDense::_inequality_jacobian = MatMapRowMajor(nullptr,0,0) [protected]

Definition at line 118 of file base_solver_nonlinear_program_dense.h.

Referenced by buildInequalityConstraintJacobian(), calculateLagrangianGradient(), teb::SolverSQPDense::calculateMeritDerivative(), and teb::SolverSQPDense::initSolverWorkspace().

16.6.5.10 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_inequality_values [protected]

Definition at line 114 of file base_solver_nonlinear_program_dense.h.

Referenced by buildInequalityConstraintJacobian(), buildInequalityConstraintValueVector(), calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMerit(), teb::SolverSQPDense::calculateMeritDerivative(), teb::SolverSQPDense::initSolverWorkspace(), and teb::SolverSQPDense::solveImpl().

16.6.5.11 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient [protected]

Definition at line 120 of file base_solver_nonlinear_program_dense.h.

Referenced by calculateLagrangianGradient(), calculateLagrangianHessianFullBFGS(), teb::SolverSQPDense::checkConvergence(), and initHessianBFGS().

16.6.5.12 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient_backup [protected]

Definition at line 121 of file base_solver_nonlinear_program_dense.h.

Referenced by calculateLagrangianHessianFullBFGS(), and initHessianBFGS().

16.6.5.13 Eigen::Matrix<double,-1,-1,Eigen::RowMajor> teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian [protected]

Definition at line 122 of file base_solver_nonlinear_program_dense.h.

Referenced by calculateLagrangianHessianFullBFGS(), calculateLagrangianHessianNumerically(), initHessianBFGS(), and teb::SolverSQPDense::solveImpl().

16.6.5.14 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_multiplier_eq [protected]

Definition at line 125 of file base_solver_nonlinear_program_dense.h.

Referenced by calculateLagrangianGradient(), calculateLagrangianHessianNumerically(), initializeLagrangeMultiplier(), and teb::SolverSQPDense::solveImpl().

16.6.5.15 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_multiplier_ineq [protected]

Definition at line 124 of file base_solver_nonlinear_program_dense.h.

Referenced by calculateLagrangianGradient(), calculateLagrangianHessianNumerically(), initializeLagrangeMultiplier(), and teb::SolverSQPDense::solveImpl().

16.6.5.16 unsigned int teb::BaseSolver::_no_vert_backups = 0 [protected], [inherited]

Definition at line 243 of file base_solver.h.

Referenced by teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::restoreVertices(), and teb::BaseSolver::solve().

16.6.5.17 int teb::BaseSolver::_objective_dim = -1 [protected], [inherited]

Definition at line 239 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::getValueDimension(), and teb::BaseSolver::solve().

16.6.5.18 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_objective_gradient [protected]

Definition at line 116 of file base_solver_nonlinear_program_dense.h.

Referenced by buildObjectiveGradient(), calculateLagrangianGradient(), teb::SolverSQPDense::calculateMerit(), teb::SolverSQPDense::calculateMeritDerivative(), and teb::SolverSQPDense::solveImpl().

16.6.5.19 double teb::BaseSolverNonlinearProgramDense::_objective_value [protected]

Definition at line 112 of file base_solver_nonlinear_program_dense.h.

Referenced by buildObjectiveValue(), and teb::SolverSQPDense::calculateMerit().

16.6.5.20 EdgeContainer* teb::BaseSolver::_objectives = nullptr [protected], [inherited]

Definition at line 234 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), buildObjectiveGradient(), buildObjectiveValue(), teb::BaseSolverLeastSquares::buildValueVector(), calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimObjectives(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), initWorkspaces(), teb::SolverNloptPackage::objectives(), and teb::-BaseSolver::solve().

16.6.5.21 int teb::BaseSolver::_opt_vec_dim = -1 [protected], [inherited]

Definition at line 238 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian(), buildEqualityConstraint-Jacobian(), buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), buildObjectiveGradient(), calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse-::countJacobianColNNZ(), teb::BaseSolver::getOptVecCopy(), teb::SolverNloptPackage::getOptVecDimFrom-Storage(), initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), teb::Solver-LevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::-SolverSQPDense::solveImpl().

16.6.5.22 const Config* teb::BaseSolver::cfg = nullptr [inherited]

See Also

setConfig()

Definition at line 95 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::build-Jacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::build-ValueVector(), calculateLagrangianHessian(), calculateLagrangianHessianFullBFGS(), initHessianBFGS(), teb::-SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::setConfig(), and teb::SolverSQPDense::solveImpl().

16.6.5.23 teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW [inherited]

Definition at line 253 of file base_solver.h.

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

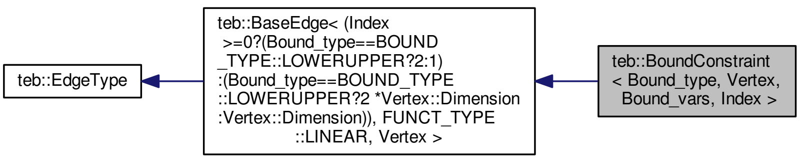
- base_solver_nonlinear_program_dense.h
- base_solver_nonlinear_program_dense.cpp

16.7 teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index > Class Template Reference

This class captures general bound constraints on optimization variables / vertices.

#include <bound_constraints.h>

Inheritance diagram for teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >:



Public Types

- using ValueVector = typename BaseEdge< NoBounds, FUNCT_TYPE::LINEAR, Vertex >::ValueVector

    Typedef to represent the static Eigen::Vector of cost/constraint values.
- using EdgeContainer = std::vector< EdgeType ∗ >

    Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).
- using ValueVectorMap = Eigen::Map< Eigen::VectorXd >

    Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.

Public Member Functions

- BoundConstraint (Vertex &vertex)

    Construct edge and attach the corresponding vertex.
- virtual const bool isBoundConstraint () const

    Override this method to specify this edge as a bound constraint since it can be handled seperately by some solvers to speed up optimization.
- virtual void computeValues ()

    Actual cost function or constraint function. For constraints it is computeValues() <= 0.
- virtual void computeJacobian ()

    Specification of the analytic block jacobian.
- virtual void computeHessian ()

    Specification of the analytic block hessian.
- void setBounds (const double ∗bounds)

    Set bounds for either BOUND_TYPE::LOWER or BOUND_TYPE::UPPER.
- void setBounds (const Eigen::Ref< const Eigen::Matrix< double, NoBounds, 1 >> &bounds)

    Set bounds for either BOUND_TYPE::LOWER or BOUND_TYPE::UPPER.
- void setBounds (double bound)

    Set single bound for either BOUND_TYPE::LOWER or BOUND_TYPE::UPPER.
- void setBounds (double lb, double ub)

    Set single bound forBOUND_TYPE::LOWERUPPER.
- void setBounds (const double ∗lb, const double ∗ub)

    Set lower and upper bounds for BOUND_TYPE::LOWERUPPER.
- void setBounds (const Eigen::Ref< const Eigen::Matrix< double, NoBounds/2, 1 >> &lb, const Eigen::Ref< const Eigen::Matrix< double, NoBounds/2, 1 >> &ub)

    Set lower and upper bounds for BOUND_TYPE::LOWERUPPER.
- const double ∗ getBounds () const

    get a constant pointer to bound data.
- virtual void ∗ getCustomData ()

    Return bound information.
- virtual int dimension () const

    Return edge dimension by function call.
- virtual FUNCT_TYPE functType () const

    Return function typed given by template parameter FuncType.
- virtual const double ∗ valuesData () const

    Return pointer to cost/constraint values [dimension() x 1] (read-only).
- virtual double ∗ valuesData ()

    Return pointer to cost/constraint values [dimension() x 1].
- virtual const ValueVector & values () const

    Return cost/constraint values as static Eigen::Vector (read-only).
- virtual ValueVector & values ()

    Return cost/constraint values as static Eigen::Vector.

- virtual void allocateMemory (bool skip_hessian=false)

  Allocate memory for JacobianWorkspace and HessianWorkspace and get vertices dimensions.
- virtual VertexType ∗ getVertex (unsigned int idx)

  Access attached vertex with index `idx`*.*
- virtual const VertexType ∗ getVertex (unsigned int idx) const

  Access attached vertex with index `idx` *(read-only).*
- virtual unsigned int noVertices () const

  Return the number of attached vertices.
- virtual void calculateVertexDimensions ()

  Return the number of attached vertices.
- virtual const std::array
  < VertexType ∗, NoVertices > & vertices () const

  Return vertex container of the edge.
- virtual ValueVectorMap valuesMap ()

  Get cost/constraints values as Eigen type matrix.
- virtual JacobianWorkspace & jacobians ()

  Access the jacobian workspace of this edge.
- virtual const JacobianWorkspace & jacobians () const

  Access the jacobian workspace of this edge (read-only).
- virtual HessianWorkspace & hessians ()

  Access the hessian workspace of this edge.
- virtual const HessianWorkspace & hessians () const

  Access the jacobian workspace of this edge (read-only).
- virtual void backupJacobian ()

  Make a backup of the current jacobian block matrix.
- virtual void restoreJacobian ()

  Restore jacobian block matrix from the backup stack.
- virtual void discardJacobianBackup ()

  Discard current jacobian backup.
- virtual JacobianWorkspace & getJacobianBackup ()

Static Public Attributes

- static const int NoBounds = Index >= 0 ? (Bound_type == BOUND_TYPE::LOWERUPPER ? 2 : 1) : (Bound_type == BOUND_TYPE::LOWERUPPER ? 2 ∗ Vertex::Dimension : Vertex::Dimension)

  Store number of bounds.
- static const BOUND_TYPE BoundType = Bound_type

  Store type of bound (lower, upper, or both);.
- static constexpr const int NoVertices = sizeof...(Vertices)
- static const int Dimension = D

  Edge dimension as static member variable.

Protected Attributes

- Eigen::Matrix< double,
  NoBounds, 1 > _bounds

  Store bounds: first lower and then upper bound, depending on Bound_type parameter.
- ValueVector _values = ValueVector::Zero()

  Actual cost/constraint value data object (initialized to zero).
- const std::array< VertexType
  ∗, NoVertices > _vertices

Container that stores all attached vertices.

- std::array< int, NoVertices > _vertices_dim

  Store dimensions for all vertices (EdgeType::_vertices, calculateVertexDimensions())
- JacobianWorkspace _jacobians

  Block-Jacobians of the edge (see JacobianWorkspace).
- HessianWorkspace _hessians

  Block-Hessians of the edge (see HessianWorkspace)
- BackupStackType
  < JacobianWorkspace > _jacob_backup

16.7.1 Detailed Description

template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1>class teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >

Use this class to create bound constraints $\mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max}$

Remarks

Call allocateMemory() before using the edge for optimization.

Todo Maybe use a special representation or zero Hessians to avoid processing a lot of zeros (same for other edges)

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

ConstraintEdgeUnary, ConstraintEdgeBinary, ConstraintEdgeMulti, ObjectiveEdgeUnary, ObjectiveEdgeBinary, ObjectiveEdgeMulti

Template Parameters

Bound_type	Lower or upper bound corresponding to BOUND_TYPE enum
Vertex	Type of the attached vertex.
Bound_vars	Specify whether a single state or all components of the VertexType should be bounded (See BOUND_VARS enum).
Index	If `Bound_vars==BOUND_VARS::SINGLE` then this Index specifies the element/component, otherwise set `Index` to -1.

Definition at line 55 of file bound_constraints.h.

16.7.2 Member Typedef Documentation

16.7.2.1 using teb::EdgeType::EdgeContainer = std::vector<EdgeType*> [inherited]

Definition at line 114 of file graph.h.

16.7.2.2 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> using teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::ValueVector = typename BaseEdge<NoBounds, FUNCT_TYPE::LINEAR, Vertex>::ValueVector

Definition at line 70 of file bound_constraints.h.

16.7.2.3 using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd > [inherited]

Definition at line 116 of file graph.h.

16.7.3 Constructor & Destructor Documentation

16.7.3.1 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::BoundConstraint (Vertex & *vertex*) [inline]

Parameters

vertex	Reference to the vertex that should be bounded

Definition at line 76 of file bound_constraints.h.

16.7.4 Member Function Documentation

16.7.4.1 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices >::allocateMemory (bool *skip_hessian* = false) [inline], [virtual], [inherited]

Remarks

Before calling, call resizeVertexContainer() edge and add all vertices (setVertex()) first.

Todo Implement automatic memory allocation instead calling this function manually.

Parameters

skip_hessian	If no hessian calculation is required by the solver skip memory allocation.

Implements teb::EdgeType.

Definition at line 84 of file base_edge.h.

16.7.4.2 virtual void teb::EdgeType::backupJacobian ( ) [inline], [virtual], [inherited]

Definition at line 162 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.7.4.3 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ( ) [inline], [virtual], [inherited]

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container EdgeType::_vertices_dim. This function is called within BaseEdge::allocateMemory().

Todo Maybe implement static/constexpr version of collecting all dimensions since they are nown at compile-time (maybe http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array-

Definition at line 110 of file base_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), and teb::BaseEdge< D, FuncType >::allocateMemory().

16.7.4.4 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual void teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian ( ) [inline], [virtual]

Reimplemented from teb::BaseEdge< D, FuncType, Vertices >.

Definition at line 163 of file bound_constraints.h.

References teb::EdgeType::_hessians, and teb::HessianWorkspace::getWorkspace().

16.7.4.5 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual void teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian ( ) [inline], [virtual]

Reimplemented from teb::BaseEdge< D, FuncType, Vertices >.

Definition at line 125 of file bound_constraints.h.

References teb::EdgeType::_jacobians, teb::JacobianWorkspace::getWorkspace(), teb::LOWER, teb::LOWERUPPER, and teb::SINGLE.

16.7.4.6 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual void teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeValues ( ) [inline], [virtual]

Reimplemented from teb::BaseEdge< D, FuncType, Vertices >.

Definition at line 85 of file bound_constraints.h.

References teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds, teb::BaseEdge< D, FuncType, Vertices >::_values, teb::BaseEdge< D, FuncType, Vertices >::_vertices, teb::LOWER, teb::LOWERUPPER, and teb::SINGLE.

16.7.4.7 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual int teb::BaseEdge< D, FuncType, Vertices >::dimension ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 51 of file base_edge.h.

16.7.4.8 virtual void teb::EdgeType::discardJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 171 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.7.4.9 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual FUNCT_TYPE teb::BaseEdge< D, FuncType, Vertices >::functType ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 69 of file base_edge.h.

16.7.4.10 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> const double* teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getBounds ( ) const [inline]

See setBounds() overloads for more information about the length. In summary:

- BOUND_TYPE::LOWER / BOUND_TYPE::UPPER && BOUND_VARS::SINGLE : 1
- BOUND_TYPE::LOWER / BOUND_TYPE::UPPER && BOUND_VARS::ALL : Vertex::Dimension

BOUND_TYPE::LOWERUPPER && BOUND_VARS::SINGLE : 2

- BOUND_TYPE::LOWERUPPER && BOUND_VARS::ALL : 2∗Vertex::Dimension

In case of BOUND_TYPE::LOWERUPPER first all lower bounds are stored and then all upper ones.

Returns

constant pointer to bound data

Definition at line 283 of file bound_constraints.h.

References teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds.

16.7.4.11 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual void∗ teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData ( ) [inline], [virtual]

This function overwrites the default interface to share custom data with edges. It avoids dynamic casting with known (!!!) template parameters. Cast the output to a CustomBoundData struct:

```
 EdgeType* edge = new BoundConstaint<>(); // Do not forget to set bounds afterwards ...
 CustomBoundData* bound_data = static_cast<CustomBoundData*>(edge->getCustomData());
// Do something with bound_data.
delete bound_data;
```

Do not forget to delete the `bound_data` object later

The bound data returned shares always the same size like `vertex->dimensionFree`.

All Bounds that are not considered in this class (like for Index>0) are set to -INF or INF depending on ther type.

Returns

Void-pointer to CustomBoundData object on the heap.

Reimplemented from teb::EdgeType.

Definition at line 308 of file bound_constraints.h.

References teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds, teb::BaseEdge< D, FuncType, Vertices >::_vertices, teb::CustomBoundData::index, INF, teb::LOWER, teb::LOWERUPPER, teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::NoBounds, and teb::UPPER.

16.7.4.12 virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 173 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.7.4.13 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual VertexType∗ teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 98 of file base_edge.h.

16.7.4.14 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 99 of file base_edge.h.

16.7.4.15 virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline], [virtual], [inherited]

Definition at line 159 of file graph.h.

References teb::EdgeType::_hessians.

16.7.4.16 virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline], [virtual], [inherited]

Definition at line 160 of file graph.h.

References teb::EdgeType::_hessians.

16.7.4.17 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual const bool teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::isBoundConstraint ( ) const [inline], [virtual]

Reimplemented from teb::EdgeType.

Definition at line 83 of file bound_constraints.h.

16.7.4.18 virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline], [virtual], [inherited]

Definition at line 157 of file graph.h.

References teb::EdgeType::_jacobians.

16.7.4.19 virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline], [virtual], [inherited]

Definition at line 158 of file graph.h.

References teb::EdgeType::_jacobians.

16.7.4.20 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual unsigned int teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 101 of file base_edge.h.

16.7.4.21 virtual void teb::EdgeType::restoreJacobian ( ) [inline], [virtual], [inherited]

Definition at line 165 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.7.4.22 template< BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1 > void teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds (const double * *bounds*) [inline]

Set an array of the bounds. The length depends on the class' template arguments BOUND_TYPE `Bound_type` and BOUND_VARS `Bound_vars` :

- BOUND_TYPE::LOWER / BOUND_TYPE::UPPER && BOUND_VARS::SINGLE : 1
- BOUND_TYPE::LOWER / BOUND_TYPE::UPPER && BOUND_VARS::ALL : Vertex::Dimension

Parameters

bounds	pointer to the bound-array.

Definition at line 182 of file bound_constraints.h.

References teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds, and teb::LOWERUPPER.

Referenced by teb::TebController< p, q >::buildOptimizationGraph().

16.7.4.23 template< BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1 > void teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds (const Eigen::Ref< const Eigen::Matrix< double, NoBounds, 1 >> & *bounds*) [inline]

Set an array of the bounds. The length depends on the class' template arguments BOUND_TYPE `Bound_type` and BOUND_VARS `Bound_vars` :

- BOUND_TYPE::LOWER / BOUND_TYPE::UPPER && BOUND_VARS::SINGLE : 1
- BOUND_TYPE::LOWER / BOUND_TYPE::UPPER && BOUND_VARS::ALL : Vertex::Dimension

Parameters

bounds	Eigen Vector containing the bound.

Definition at line 196 of file bound_constraints.h.

References teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds, and teb::LOWERUPPER.

16.7.4.24 template< BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1 > void teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds (double *bound*) [inline]

Set a single bound. BOUND_VARS::SINGLE and `Index>=0` template parameter is required!

Parameters

bound	bound-value

Definition at line 207 of file bound_constraints.h.

References teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds, teb::LOWERUPPER, and teb::SINGLE.

16.7.4.25 template< BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1 > void teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds (double *lb,* double *ub*) [inline]

Set a single bound. BOUND_VARS::SINGLE and `Index>=0` template parameter are required!

Parameters

lb	lower bound-value
ub	upper bound-value

Definition at line 219 of file bound_constraints.h.

References teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds, teb::LOWERUPPER, and teb::SINGLE.

16.7.4.26 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> void teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds (const double ∗ *lb*, const double ∗ *ub*) [inline]

Set an array of the bounds. The length depends on the class' template arguments BOUND_TYPE `Bound_type` and BOUND_VARS `Bound_vars` :

- BOUND_TYPE::LOWERUPPER && BOUND_VARS::SINGLE : 2
- BOUND_TYPE::LOWERUPPER && BOUND_VARS::ALL : 2∗Vertex::Dimension

  Parameters

lb	pointer to the lower bound-array.
ub	pointer to the upper bound-array.

Definition at line 235 of file bound_constraints.h.

References teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds, teb::LOWERUPPER, and teb::SINGLE.

16.7.4.27 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> void teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds (const Eigen::Ref< const Eigen::Matrix< double, NoBounds/2, 1 >> & *lb*, const Eigen::Ref< const Eigen::Matrix< double, NoBounds/2, 1 >> & *ub*) [inline]

Set an array of the bounds. The length depends on the class' template arguments BOUND_TYPE `Bound_type` and BOUND_VARS `Bound_vars` :

- BOUND_TYPE::LOWERUPPER && BOUND_VARS::SINGLE : 2
- BOUND_TYPE::LOWERUPPER && BOUND_VARS::ALL : 2∗Vertex::Dimension

  Parameters

lb	Eigen::Vector [NoBounds x 1] for the lower bound.
ub	Eigen::Vector [NoBounds x 1] for the upper bound.

Definition at line 262 of file bound_constraints.h.

References teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds, teb::LOWERUPPER, and teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::NoBounds.

16.7.4.28 template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) const [inline], [virtual], [inherited]

Definition at line 74 of file base_edge.h.

16.7.4.29 template< int D, FUNCT_TYPE FuncType, class... Vertices> virtual ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) [inline], [virtual], [inherited]

Definition at line 75 of file base_edge.h.

16.7.4.30 template< int D, FUNCT_TYPE FuncType, class... Vertices> virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 72 of file base_edge.h.

16.7.4.31 template< int D, FUNCT_TYPE FuncType, class... Vertices> virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 73 of file base_edge.h.

16.7.4.32 virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline], [virtual], [inherited]

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References teb::EdgeType::dimension(), and teb::EdgeType::valuesData().

16.7.4.33 template< int D, FUNCT_TYPE FuncType, class... Vertices> virtual const std::array< VertexType*,NoVertices>& teb::BaseEdge< D, FuncType, Vertices >::vertices ( ) const [inline], [virtual], [inherited]

Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file base_edge.h.

16.7.5 Member Data Documentation

16.7.5.1 template< BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> Eigen::Matrix<double, NoBounds, 1> teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds [protected]

Definition at line 377 of file bound_constraints.h.

Referenced by teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeValues(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getBounds(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData(), and teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds().

16.7.5.2 HessianWorkspace teb::EdgeType::_hessians [protected], [inherited]

Definition at line 178 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeMinimizeTime::computeHessian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), and teb::EdgeType::hessians().

16.7.5.3 BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup [protected], [inherited]

Definition at line 180 of file graph.h.

Referenced by teb::EdgeType::backupJacobian(), teb::EdgeType::discardJacobianBackup(), teb::EdgeType::getJacobianBackup(), and teb::EdgeType::restoreJacobian().

16.7.5.4 JacobianWorkspace teb::EdgeType::_jacobians [protected], [inherited]

Definition at line 173 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeType::backupJacobian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::EdgeMinimizeTime::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::EdgeType::jacobians(), and teb::EdgeType::restoreJacobian().

16.7.5.5 template<int D, FUNCT_TYPE FuncType, class... Vertices> ValueVector teb::BaseEdge< D, FuncType, Vertices >::_values = ValueVector::Zero() [protected], [inherited]

Definition at line 139 of file base_edge.h.

Referenced by teb::BaseEdge< D, FuncType >::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeValues(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::values(), teb::BaseEdge< D, FuncType >::values(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::valuesData(), and teb::BaseEdge< D, FuncType >::valuesData().

16.7.5.6 template<int D, FUNCT_TYPE FuncType, class... Vertices> const std::array<VertexType*, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices [protected], [inherited]

Definition at line 141 of file base_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::calculateVertexDimensions(), teb::BaseEdge< D, FuncType >::calculateVertexDimensions(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeValues(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::getVertex(), teb::BaseEdge< D, FuncType >::getVertex(), teb::BaseEdge< D, FuncType >::noVertices(), teb::BaseEdge< D, FuncType >::resizeVertexContainer(), teb::BaseEdge< D, FuncType >::setVertex(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::vertices(), and teb::BaseEdge< D, FuncType >::vertices().

16.7.5.7 template<int D, FUNCT_TYPE FuncType, class... Vertices> std::array<int, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim [protected], [inherited]

Definition at line 142 of file base_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::calculateVertexDimensions(), and teb::BaseEdge< D, FuncType >::calculateVertexDimensions().

16.7.5.8 template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> const BOUND_TYPE teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::BoundType = Bound_type [static]

Definition at line 66 of file bound_constraints.h.

16.7.5.9 template< int D, FUNCT_TYPE FuncType, class... Vertices > const int teb::BaseEdge< D, FuncType, Vertices >::Dimension = D [static], [inherited]

Definition at line 50 of file base_edge.h.

16.7.5.10 template< BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1 > const int teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::NoBounds = Index >= 0 ? (Bound_type == BOUND_TYPE::LOWERUPPER ? 2 : 1) : (Bound_type == BOUND_TYPE::LOWERUPPER ? 2 * Vertex::Dimension : Vertex::Dimension) [static]

Definition at line 65 of file bound_constraints.h.

Referenced by teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData(), and teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds().

16.7.5.11 template< int D, FUNCT_TYPE FuncType, class... Vertices > constexpr const int teb::BaseEdge< D, FuncType, Vertices >::NoVertices = sizeof...(Vertices) [static], [inherited]

Definition at line 48 of file base_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::noVertices().

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

- bound_constraints.h

16.8 teb::Config Class Reference

Configurations for Controller and Solver classes.

`#include <config.h>`

Classes

- struct Optim

  Configurations related to the trajectory optimization.
- struct Teb

  Configurations related to the TEB algorithm.
- struct Utilities

  Configurations for helper miscellaneous and utilities.

Public Attributes

- struct teb::Config::Teb teb

  Configurations related to the TEB algorithm.
- struct teb::Config::Optim optim

  Configurations related to the trajectory optimization.
- struct teb::Config::Utilities util

  Configurations for helper miscellaneous and utilities.

16.8.1 Detailed Description

This class defines the main config for controller and solver classes.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

BaseController TebController BaseSolver

Examples:

integrator_system1.cpp, integrator_system_classic_mpc.cpp, linear_system_ode_ctrl_comparison.cpp, mobile_robot_teb.cpp, and rocket_system.cpp.

Definition at line 55 of file config.h.

16.8.2 Member Data Documentation

16.8.2.1 struct teb::Config::Optim teb::Config::optim

Examples:

integrator_system_classic_mpc.cpp, linear_system_ode_ctrl_comparison.cpp, mobile_robot_teb.cpp, and rocket_system.cpp.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), and teb::SolverSQPDense::solveImpl().

16.8.2.2 struct teb::Config::Teb teb::Config::teb

Examples:

mobile_robot_teb.cpp.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::TebController< p, q >::resetController(), and teb::TebController< p, q >::setupHorizon().

16.8.2.3 struct teb::Config::Utilities teb::Config::util

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

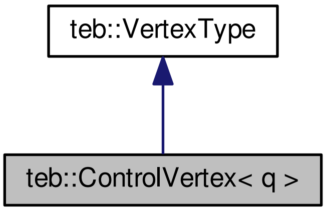
- config.h

16.9 teb::ControlVertex< q > Class Template Reference

Vertex that contains the ControlVector.

```
#include <teb_vertices.h>
```

Inheritance diagram for teb::ControlVertex< q >:



Public Types

- using ControlVector = Eigen::Matrix< double, q, 1 >

  Typedef for control input vector with q controls [q x 1].
- using ScalarType = Eigen::Matrix< double, 1, 1 >

  Typedef for a 1D-scalar value represented as Eigen::Matrix type [1 x 1].
- using FixedCtrls = Eigen::Matrix< bool, q, 1 >

  Typedef for logical mask that stores fixed and unfixed controls [q x 1].
- using BackupStackType = std::stack< ControlVector, std::vector< ControlVector, Eigen::aligned_allocator< ControlVector > > >

  Typdef for the backup stack.
- using VertexContainer = std::vector< VertexType ∗ >

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).

Public Member Functions

- ControlVertex ()

  Default constructor: control vector initialized to zero.
- ControlVertex (const Eigen::Ref< const ControlVector > &controls)

  Construct ControlVertex with given control inputs.
- ControlVertex (const ControlVertex< q > &obj)

  Copy constructor.
- virtual int dimension () const

  Return number of elements/values/components stored in this vertex.
- virtual int dimensionFree () const

  Return only the dimension of free (unfixed) variables.
- virtual void plus (const VertexType ∗rhs)

  Define the increment for the vertex: x = x + rhs.
- virtual void plus (const double ∗rhs)

  Define the increment for the vertex: x = x + rhs (rhs[] with dimension()=p+q values)
- virtual void plusFree (const double ∗rhs)

  Define the increment for the vertex: x = x + rhs (But only add FREE variables, rhs[] with dimensionFree() values).
- virtual void setFree (const double ∗rhs)

  Overwrite all free variables with the values given by rhs (rhs[] with dimensionFree() values).
- virtual void getDataFree (double ∗target_vec) const

Return a copy of the free (unfixed) values as a single array (length: dimensionFree()).

- virtual const double & getData (unsigned int idx) const

  Return pointer to data with the index `idx`).
- const ControlVector & controls () const

  Get ControlVector (read-only).
- ControlVector & controls ()

  Get ControlVector.
- virtual void setControls (const Eigen::Ref< const ControlVector > &c)

  Set or update ControlVector.
- virtual void setControls (double control)

  Set or update only first component of the ControlVector (useful for 1d controls).
- const FixedCtrls & fixed_ctrls () const

  Get logical map that defines fixed and unfixed controls (fixed = `true`).
- void setFixedAll (bool fixed)

  Set all controls of this object fixed or unfixed.
- void setFixedCtrl (unsigned int ctrl_idx, bool fixed)

  Set a single component of the ControlVector to fixed/unfixed.
- void setFixedCtrls (bool fixed)

  Set all controls of the ControlVector to fixed/unfixed.
- void setFixedCtrls (const Eigen::Ref< const FixedCtrls > &fixed_ctrls)

  Set selected controls of the ControlVector to fixed/unfixed at once (Eigen version).
- void setFixedCtrls (const bool *fixed)

  Set selected components of the ControlVector to fixed/unfixed at once.
- virtual bool isFixedAll () const

  return `true`, if ALL values of this vertex are fixed and therefore are not necessary for optimization.
- bool isFixedAny () const

  return `true`, if ANY element/value/component of this vertex is fixed and therefore the vertex is relevant for optimization.
- virtual bool isFixedComp (int idx) const

  return `true`, if element/value/component with index `idx` of this vertex is fixed.
- virtual void push ()

  This function should store all values into a internal backup stack.
- virtual void pop ()

  This function should restore the previously stored values of the backup stack and remove them from the stack.
- virtual void top ()

  This function should restore the previously stored values of the backup stack WITHOUT removing them from the stack.
- virtual void discardTop ()

  This function should delete the previously made backup from the stack without restoring it.
- virtual unsigned int stackSize () const

  This function should return the current size/number of backups of the backup stack.
- ControlVertex< q > & operator= (const ControlVertex< q > &rhs)
- void setOptVecIdx (int opt_vec_idx)

  Update the position of the first value of this vertex inside the optimization vector / Hessian.
- int getOptVecIdx () const

  Get the position of the first value of this Vertex inside the optimization vector / Hessian.

Static Public Attributes

- static const int DimControls = q

  Number of control inputs / Size of the control vector as static variable.
- static const int Dimension = q

  Number of total values stored in this vertex (number of control input vector components).

Protected Attributes

- ControlVector _controls
  Control input vector with q controls [q x 1].
- BackupStackType _backup
  backup stack (store and restore control input).
- int _dimension = q
  Store dimension for all values stored (q controls)
- FixedCtrls _fixed_ctrls = FixedCtrls::Constant(false)
  Mask that stores fixed and unfixed controls [q x 1] true: fixed, false: free.
- int _opt_vec_idx = -1
  Start-index in optimization vector (idx in hessian (row/column) and jacobian (column))

Friends

- bool operator== (const ControlVertex< q > &lhs, const ControlVertex< q > &rhs)
- bool operator!= (const ControlVertex< q > &lhs, const ControlVertex< q > &rhs)
- std::ostream & operator<< (std::ostream &os, const ControlVertex< q > &control)
  Print control vector using std::ostream.

16.9.1 Detailed Description

template< int q>class teb::ControlVertex< q >

This class wraps the discrete control input vector $\mathbf{u}_k \in \mathbb{R}^q$ into a vertex for the hyper-graph.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

StateVertex, VertexType, TimeDiff, BaseEdge

Template Parameters

q	Number of control inputs / Size of the control vector

Definition at line 281 of file teb_vertices.h.

16.9.2 Member Typedef Documentation

16.9.2.1 template< int q> using teb::ControlVertex< q >::BackupStackType = std::stack< ControlVector, std::vector< ControlVector, Eigen::aligned_allocator< ControlVector > > >

Definition at line 298 of file teb_vertices.h.

16.9.2.2 template< int q> using teb::ControlVertex< q >::ControlVector = Eigen::Matrix< double, q, 1 >

Definition at line 288 of file teb_vertices.h.

16.9.2.3 template< int q> using teb::ControlVertex< q >::FixedCtrls = Eigen::Matrix< bool, q, 1 >

Definition at line 292 of file teb_vertices.h.

16.9.2.4 template<int q> using teb::ControlVertex< q >::ScalarType = Eigen::Matrix<double, 1, 1>

Definition at line 290 of file teb_vertices.h.

16.9.2.5 using teb::VertexType::VertexContainer = std::vector<VertexType*> [inherited]

Definition at line 33 of file graph.h.

16.9.3 Constructor & Destructor Documentation

16.9.3.1 template<int q> teb::ControlVertex< q >::ControlVertex ( ) [inline]

Definition at line 306 of file teb_vertices.h.

16.9.3.2 template<int q> teb::ControlVertex< q >::ControlVertex (const Eigen::Ref< const ControlVector > & *controls*) [inline]

Parameters

controls	Eigen::Vector with `q` control inputs.

Definition at line 312 of file teb_vertices.h.

16.9.3.3 template<int q> teb::ControlVertex< q >::ControlVertex (const ControlVertex< q > & *obj*) [inline]

Parameters

obj	Object of type ControlVertex

Definition at line 318 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls, and teb::ControlVertex< q >::fixed_ctrls().

16.9.4 Member Function Documentation

16.9.4.1 template<int q> const ControlVector& teb::ControlVertex< q >::controls ( ) const [inline]

Returns

control vector [q x 1]

Definition at line 397 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls.

Referenced by teb::ControlVertex< q >::operator=(), and teb::ControlVertex< q >::plus().

16.9.4.2 template<int q> ControlVector& teb::ControlVertex< q >::controls ( ) [inline]

Returns

control vector [q x 1]

Definition at line 398 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls.

16.9.4.3 template< int q > virtual int teb::ControlVertex< q >::dimension ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 327 of file teb_vertices.h.

16.9.4.4 template< int q > virtual int teb::ControlVertex< q >::dimensionFree ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 329 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls.

16.9.4.5 template< int q > virtual void teb::ControlVertex< q >::discardTop ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 456 of file teb_vertices.h.

References teb::ControlVertex< q >::_backup.

16.9.4.6 template< int q > const FixedCtrls& teb::ControlVertex< q >::fixed_ctrls ( ) const [inline]

Returns

logical map (q x 1 vector of booleans)

Definition at line 401 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls.

Referenced by teb::ControlVertex< q >::ControlVertex(), teb::ControlVertex< q >::operator=(), and teb::ControlVertex< q >::setFixedCtrls().

16.9.4.7 template< int q > virtual const double& teb::ControlVertex< q >::getData (unsigned int *idx*) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 388 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls.

16.9.4.8 template< int q > virtual void teb::ControlVertex< q >::getDataFree (double * *target_vec*) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 375 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls, and teb::ControlVertex< q >::isFixedComp().

16.9.4.9 int teb::VertexType::getOptVecIdx ( ) const [inline], [inherited]

Returns

position of the first element of this vertex.

Definition at line 59 of file graph.h.

References teb::VertexType::_opt_vec_idx.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), and teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ().

16.9.4.10 template< int q> virtual bool **teb::ControlVertex< q >::isFixedAll** ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 438 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls.

16.9.4.11 template< int q> bool **teb::ControlVertex< q >::isFixedAny** ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 439 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls.

16.9.4.12 template< int q> virtual bool **teb::ControlVertex< q >::isFixedComp** (int *idx*) const [inline], [virtual]

Parameters

idx	component index

Implements teb::VertexType.

Definition at line 441 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls.

Referenced by teb::ControlVertex< q >::getDataFree(), teb::ControlVertex< q >::plusFree(), and teb::ControlVertex< q >::setFree().

16.9.4.13 template< int q> **ControlVertex< q>& teb::ControlVertex< q >::operator=** (const **ControlVertex< q >** & *rhs*) [inline]

Definition at line 461 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls, teb::ControlVertex< q >::_fixed_ctrls, teb::ControlVertex< q >::controls(), and teb::ControlVertex< q >::fixed_ctrls().

16.9.4.14 template< int q> virtual void **teb::ControlVertex< q >::plus** (const **VertexType** * *rhs*) [inline], [virtual]

Implements teb::VertexType.

Definition at line 332 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls, and teb::ControlVertex< q >::controls().

16.9.4.15 template<int q> virtual void teb::ControlVertex< q >::plus (const double * rhs) [inline], [virtual]

Implements teb::VertexType.

Definition at line 339 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls.

16.9.4.16 template<int q> virtual void teb::ControlVertex< q >::plusFree (const double * rhs) [inline], [virtual]

Implements teb::VertexType.

Definition at line 345 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls, and teb::ControlVertex< q >::isFixedComp().

16.9.4.17 template<int q> virtual void teb::ControlVertex< q >::pop ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 446 of file teb_vertices.h.

References teb::ControlVertex< q >::_backup, and teb::ControlVertex< q >::top().

16.9.4.18 template<int q> virtual void teb::ControlVertex< q >::push ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 445 of file teb_vertices.h.

References teb::ControlVertex< q >::_backup, and teb::ControlVertex< q >::_controls.

16.9.4.19 template<int q> virtual void teb::ControlVertex< q >::setControls (const Eigen::Ref< const ControlVector > & c) [inline], [virtual]

Parameters

c	control vector [p x 1]

Definition at line 399 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls.

16.9.4.20 template<int q> virtual void teb::ControlVertex< q >::setControls (double control) [inline], [virtual]

Parameters

control	single control component.

Definition at line 400 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls.

16.9.4.21 template<int q> void teb::ControlVertex< q >::setFixedAll (bool fixed) [inline]

Parameters

fixed	set to `true` in order to fix all *q* controls.

Definition at line 407 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls.

16.9.4.22 template<int q> void **teb::ControlVertex**< q >::setFixedCtrl (unsigned int *ctrl_idx*, bool *fixed*) [inline]

Parameters

ctrl_idx	index of the underlying ControlVector component that should be fixed.
fixed	set to `true` in order to fix the selcted control.

Definition at line 418 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls.

16.9.4.23 template<int q> void **teb::ControlVertex**< q >::setFixedCtrls (bool *fixed*) [inline]

Parameters

fixed	set to `true` in order to fix all *q* controls.

Definition at line 424 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls.

16.9.4.24 template<int q> void **teb::ControlVertex**< q >::setFixedCtrls (const Eigen::Ref< const **FixedCtrls** > & *fixed_ctrls*) [inline]

Parameters

fixed_ctrls	Eigen::Vector with *q* booleans in which each `true` sets the corresponding control of the ControlVector to fixed.

Definition at line 430 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls, and teb::ControlVertex< q >::fixed_ctrls().

16.9.4.25 template<int q> void **teb::ControlVertex**< q >::setFixedCtrls (const bool * *fixed*) [inline]

Parameters

fixed	boolean array with *q* elements in which each `true` sets the corresponding control of the ControlVector to fixed.

Definition at line 436 of file teb_vertices.h.

References teb::ControlVertex< q >::_fixed_ctrls.

16.9.4.26 template<int q> virtual void **teb::ControlVertex**< q >::setFree (const double * *rhs*) [inline], [virtual]

Implements teb::VertexType.

Definition at line 360 of file teb_vertices.h.

References teb::ControlVertex< q >::_controls, and teb::ControlVertex< q >::isFixedComp().

16.9.4.27 void teb::VertexType::setOptVecIdx (int *opt_vec_idx*) [inline], [inherited]

Parameters

opt_vec_idx	index of the first value

Definition at line 53 of file graph.h.

References teb::VertexType::_opt_vec_idx.

16.9.4.28 template<int q> virtual unsigned int teb::ControlVertex< q >::stackSize ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 457 of file teb_vertices.h.

References teb::ControlVertex< q >::_backup.

16.9.4.29 template<int q> virtual void teb::ControlVertex< q >::top ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 451 of file teb_vertices.h.

References teb::ControlVertex< q >::_backup, and teb::ControlVertex< q >::_controls.

Referenced by teb::ControlVertex< q >::pop().

16.9.5 Friends And Related Function Documentation

16.9.5.1 template<int q> bool operator!= (const ControlVertex< q > & *lhs*, const ControlVertex< q > & *rhs*) [friend]

Definition at line 479 of file teb_vertices.h.

16.9.5.2 template<int q> std::ostream& operator<< (std::ostream & *os*, const ControlVertex< q > & *control*) [friend]

Parameters

os	output stream object
control	specific ControlVertex object

Returns

output stream object including a formatted string containing control input vector.

Definition at line 489 of file teb_vertices.h.

16.9.5.3 template<int q> bool operator== (const ControlVertex< q > & *lhs*, const ControlVertex< q > & *rhs*) [friend]

Definition at line 474 of file teb_vertices.h.

16.9.6 Member Data Documentation

16.9.6.1 template<int q> BackupStackType teb::ControlVertex< q >::_backup [protected]

Definition at line 497 of file teb_vertices.h.

Referenced by teb::ControlVertex< q >::discardTop(), teb::ControlVertex< q >::pop(), teb::ControlVertex< q >::push(), teb::ControlVertex< q >::stackSize(), and teb::ControlVertex< q >::top().

16.9.6.2 template<int q> ControlVector teb::ControlVertex< q >::_controls [protected]

Definition at line 496 of file teb_vertices.h.

Referenced by teb::ControlVertex< q >::controls(), teb::ControlVertex< q >::getData(), teb::ControlVertex< q >::getDataFree(), teb::ControlVertex< q >::operator=(), teb::ControlVertex< q >::plus(), teb::ControlVertex< q >::plusFree(), teb::ControlVertex< q >::push(), teb::ControlVertex< q >::setControls(), teb::ControlVertex< q >::setFree(), and teb::ControlVertex< q >::top().

16.9.6.3 template<int q> int teb::ControlVertex< q >::_dimension = q [protected]

Definition at line 499 of file teb_vertices.h.

16.9.6.4 template<int q> FixedCtrls teb::ControlVertex< q >::_fixed_ctrls = FixedCtrls::Constant(false) [protected]

Definition at line 500 of file teb_vertices.h.

Referenced by teb::ControlVertex< q >::ControlVertex(), teb::ControlVertex< q >::dimensionFree(), teb::ControlVertex< q >::fixed_ctrls(), teb::ControlVertex< q >::isFixedAll(), teb::ControlVertex< q >::isFixedAny(), teb::ControlVertex< q >::isFixedComp(), teb::ControlVertex< q >::operator=(), teb::ControlVertex< q >::setFixedAll(), teb::ControlVertex< q >::setFixedCtrl(), and teb::ControlVertex< q >::setFixedCtrls().

16.9.6.5 int teb::VertexType::_opt_vec_idx = -1 [protected], [inherited]

Definition at line 69 of file graph.h.

Referenced by teb::VertexType::getOptVecIdx(), and teb::VertexType::setOptVecIdx().

16.9.6.6 template<int q> const int teb::ControlVertex< q >::DimControls = q [static]

Definition at line 284 of file teb_vertices.h.

16.9.6.7 template<int q> const int teb::ControlVertex< q >::Dimension = q [static]

Definition at line 285 of file teb_vertices.h.

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

- teb_vertices.h

16.10 teb::CustomBoundData Struct Reference

Datastruct for custom bound data that can be queried from the BoundConstraint class.

`#include <bound_constraints.h>`

Public Attributes

- int index = -1

Index for the bound (-1 for multiple components)

- int no_lower = 0

    Number of lower bounds.

- Eigen::VectorXd lower

    [no_lower x 1] lower bounds vector

- int no_upper = 0

    Number of upper bounds.

- Eigen::VectorXd upper

    [no_upper x 1] upper bounds vector

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW

16.10.1 Detailed Description

Definition at line 23 of file bound_constraints.h.

16.10.2 Member Data Documentation

16.10.2.1 teb::CustomBoundData::EIGEN_MAKE_ALIGNED_OPERATOR_NEW

Definition at line 31 of file bound_constraints.h.

16.10.2.2 int teb::CustomBoundData::index = -1

Definition at line 25 of file bound_constraints.h.

Referenced by teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData().

16.10.2.3 Eigen::VectorXd teb::CustomBoundData::lower

Definition at line 27 of file bound_constraints.h.

16.10.2.4 int teb::CustomBoundData::no_lower = 0

Definition at line 26 of file bound_constraints.h.

16.10.2.5 int teb::CustomBoundData::no_upper = 0

Definition at line 28 of file bound_constraints.h.

16.10.2.6 Eigen::VectorXd teb::CustomBoundData::upper

Definition at line 29 of file bound_constraints.h.

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

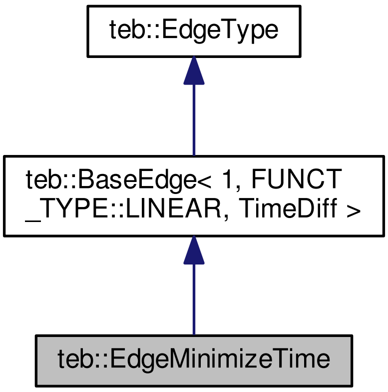
- bound_constraints.h

16.11 teb::EdgeMinimizeTime Class Reference

Objective edge: minimize ΔT.

`#include <common_teb_edges.h>`

Inheritance diagram for teb::EdgeMinimizeTime:



Public Types

- using ValueVector = Eigen::Matrix< double, D, 1 >

  Typedef to represent the static Eigen::Vector of cost/constraint values.
- using EdgeContainer = std::vector< EdgeType ∗ >

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).
- using ValueVectorMap = Eigen::Map< Eigen::VectorXd >

  Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.

Public Member Functions

- EdgeMinimizeTime (TimeDiff &dt_vertex)

  Construct edge and attach the corresponding TimeDiff vertex.
- virtual void computeValues ()

  Actual cost function or constraint function. For constraints it is computeValues() <= 0.
- virtual void computeJacobian ()

  Specification of the analytic block jacobian.
- virtual void computeHessian ()

  Specification of the analytic block hessian.
- void setData (double data)

  Set weight σ for the cost function.
- virtual int dimension () const

  Return edge dimension by function call.
- virtual FUNCT_TYPE functType () const

  Return function typed given by template parameter FuncType.
- virtual const double ∗ valuesData () const

Return pointer to cost/constraint values [dimension() x 1] (read-only).

- virtual double ∗ valuesData ()

  Return pointer to cost/constraint values [dimension() x 1].

- virtual const ValueVector & values () const

  Return cost/constraint values as static Eigen::Vector (read-only).

- virtual ValueVector & values ()

  Return cost/constraint values as static Eigen::Vector.

- virtual void allocateMemory (bool skip_hessian=false)

  Allocate memory for JacobianWorkspace and HessianWorkspace and get vertices dimensions.

- virtual VertexType ∗ getVertex (unsigned int idx)

  Access attached vertex with index `idx`*.*

- virtual const VertexType ∗ getVertex (unsigned int idx) const

  Access attached vertex with index `idx` *(read-only).*

- virtual unsigned int noVertices () const

  Return the number of attached vertices.

- virtual void calculateVertexDimensions ()

  Return the number of attached vertices.

- virtual const std::array
  < VertexType ∗, NoVertices > & vertices () const

  Return vertex container of the edge.

- virtual const bool isBoundConstraint () const

  Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.

- virtual ValueVectorMap valuesMap ()

  Get cost/constraints values as Eigen type matrix.

- virtual void ∗ getCustomData ()

  Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.

- virtual JacobianWorkspace & jacobians ()

  Access the jacobian workspace of this edge.

- virtual const JacobianWorkspace & jacobians () const

  Access the jacobian workspace of this edge (read-only).

- virtual HessianWorkspace & hessians ()

  Access the hessian workspace of this edge.

- virtual const HessianWorkspace & hessians () const

  Access the jacobian workspace of this edge (read-only).

- virtual void backupJacobian ()

  Make a backup of the current jacobian block matrix.

- virtual void restoreJacobian ()

  Restore jacobian block matrix from the backup stack.

- virtual void discardJacobianBackup ()

  Discard current jacobian backup.

- virtual JacobianWorkspace & getJacobianBackup ()

Static Public Attributes

- static constexpr const int NoVertices
- static const int Dimension

  Edge dimension as static member variable.

Protected Attributes

- double _data = 1
  Weight of the objective function.
- ValueVector _values
  Actual cost/constraint value data object (initialized to zero).
- const std::array< VertexType *, NoVertices > _vertices
  Container that stores all attached vertices.
- std::array< int, NoVertices > _vertices_dim
  Store dimensions for all vertices (EdgeType::_vertices, calculateVertexDimensions())
- JacobianWorkspace _jacobians
  Block-Jacobians of the edge (see JacobianWorkspace).
- HessianWorkspace _hessians
  Block-Hessians of the edge (see HessianWorkspace)
- BackupStackType < JacobianWorkspace > _jacob_backup

16.11.1 Detailed Description

The following cost function is implemented:

$$v = \sigma \Delta T$$

σ denotes the weight. Choosing (n -1) has performed fine in the past, since $T = (n-1)\Delta T$.

Set the weight σ using setData().

Vertices required (connect the edge with the following vertices):

1. TimeDiff ΔT

Remarks

Call allocateMemory() before using the edge for optimization.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Definition at line 32 of file common_teb_edges.h.

16.11.2 Member Typedef Documentation

16.11.2.1 using teb::EdgeType::EdgeContainer = std::vector<EdgeType*> [inherited]

Definition at line 114 of file graph.h.

16.11.2.2 using teb::BaseEdge< D, FuncType, Vertices >::ValueVector = Eigen::Matrix<double, D, 1> [inherited]

Definition at line 54 of file base_edge.h.

16.11.2.3 using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd > [inherited]

Definition at line 116 of file graph.h.

16.11.3 Constructor & Destructor Documentation

16.11.3.1 teb::EdgeMinimizeTime::EdgeMinimizeTime (TimeDiff & *dt_vertex*) [inline]

Parameters

dt_vertex	Reference to the TimeDiff vertex

Definition at line 40 of file common_teb_edges.h.

16.11.4 Member Function Documentation

16.11.4.1 virtual void teb::BaseEdge< D, FuncType, Vertices >::allocateMemory (bool *skip_hessian* = false) [inline], [virtual], [inherited]

Remarks

Before calling, call resizeVertexContainer() edge and add all vertices (setVertex()) first.

Todo Implement automatic memory allocation instead calling this function manually.

Parameters

skip_hessian	If no hessian calculation is required by the solver skip memory allocation.

Implements teb::EdgeType.

Definition at line 84 of file base_edge.h.

References teb::EdgeType::_hessians, teb::EdgeType::_jacobians, teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim, teb::JacobianWorkspace::allocate(), teb::HessianWorkspace::allocate(), and teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions().

16.11.4.2 virtual void teb::EdgeType::backupJacobian ( ) [inline], [virtual], [inherited]

Definition at line 162 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.11.4.3 virtual void teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ( ) [inline], [virtual], [inherited]

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container EdgeType::_vertices_dim. This function is called within BaseEdge::allocateMemory().

Todo Maybe implement static/constexpr version of collecting all dimensions since they are nown at compile-time (maybe http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array-

Definition at line 110 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices, and teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim.

16.11.4.4 virtual void teb::EdgeMinimizeTime::computeHessian ( ) [inline], [virtual]

Reimplemented from teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

Definition at line 59 of file common_teb_edges.h.

References teb::EdgeType::_hessians, and teb::HessianWorkspace::getWorkspace().

16.11.4.5 virtual void teb::EdgeMinimizeTime::computeJacobian ( ) [inline], [virtual]

Reimplemented from teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

Definition at line 52 of file common_teb_edges.h.

References _data, teb::EdgeType::_jacobians, and teb::JacobianWorkspace::getWorkspace().

16.11.4.6 virtual void teb::EdgeMinimizeTime::computeValues ( ) [inline], [virtual]

Reimplemented from teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

Definition at line 45 of file common_teb_edges.h.

References _data, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::_values, teb::BaseEdge< 1, FUNCT-_TYPE::LINEAR, TimeDiff >::_vertices, and teb::TimeDiff::dt().

16.11.4.7 virtual int teb::BaseEdge< D, FuncType, Vertices >::dimension ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 51 of file base_edge.h.

16.11.4.8 virtual void teb::EdgeType::discardJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 171 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.11.4.9 virtual FUNCT_TYPE teb::BaseEdge< D, FuncType, Vertices >::functType ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 69 of file base_edge.h.

16.11.4.10 virtual void* teb::EdgeType::getCustomData ( ) [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 154 of file graph.h.

16.11.4.11 virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 173 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.11.4.12 virtual VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 98 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.11.4.13 virtual const VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 99 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.11.4.14 virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline], [virtual], [inherited]

Definition at line 159 of file graph.h.

References teb::EdgeType::_hessians.

16.11.4.15 virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline], [virtual], [inherited]

Definition at line 160 of file graph.h.

References teb::EdgeType::_hessians.

16.11.4.16 virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 124 of file graph.h.

16.11.4.17 virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline], [virtual], [inherited]

Definition at line 157 of file graph.h.

References teb::EdgeType::_jacobians.

16.11.4.18 virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline], [virtual], [inherited]

Definition at line 158 of file graph.h.

References teb::EdgeType::_jacobians.

16.11.4.19 virtual unsigned int teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 101 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::NoVertices.

16.11.4.20 virtual void teb::EdgeType::restoreJacobian ( ) [inline], [virtual], [inherited]

Definition at line 165 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.11.4.21 void teb::EdgeMinimizeTime::setData (double *data*) [inline]

Definition at line 64 of file common_teb_edges.h.

References _data.

Referenced by teb::TebController< p, q >::buildOptimizationGraph().

16.11.4.22 virtual const ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) const [inline], [virtual], [inherited]

Definition at line 74 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_values.

16.11.4.23 virtual ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) [inline], [virtual], [inherited]

Definition at line 75 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_values.

16.11.4.24 virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 72 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_values.

16.11.4.25 virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 73 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_values.

16.11.4.26 virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline], [virtual], [inherited]

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References teb::EdgeType::dimension(), and teb::EdgeType::valuesData().

16.11.4.27 virtual const std::array<VertexType*,NoVertices>& teb::BaseEdge< D, FuncType, Vertices >::vertices ( ) const [inline], [virtual], [inherited]

Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file base_edge.h.

References teb::BaseEdge< D, FuncType, Vertices >::_vertices.

16.11.5 Member Data Documentation

16.11.5.1 double teb::EdgeMinimizeTime::_data = 1 [protected]

Definition at line 67 of file common_teb_edges.h.

Referenced by computeJacobian(), computeValues(), and setData().

16.11.5.2 HessianWorkspace teb::EdgeType::_hessians [protected], [inherited]

Definition at line 178 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), computeHessian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), and teb::EdgeType::hessians().

16.11.5.3 BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup [protected], [inherited]

Definition at line 180 of file graph.h.

Referenced by teb::EdgeType::backupJacobian(), teb::EdgeType::discardJacobianBackup(), teb::EdgeType::getJacobianBackup(), and teb::EdgeType::restoreJacobian().

16.11.5.4 JacobianWorkspace teb::EdgeType::_jacobians [protected], [inherited]

Definition at line 173 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeType::backupJacobian(), teb::BaseEdge< D, FuncType >::computeHessian(), computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::EdgeType::jacobians(), and teb::EdgeType::restoreJacobian().

16.11.5.5 ValueVector teb::BaseEdge< D, FuncType, Vertices >::_values [protected], [inherited]

Definition at line 139 of file base_edge.h.

Referenced by computeValues().

16.11.5.6 const std::array<VertexType*, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices [protected], [inherited]

Definition at line 141 of file base_edge.h.

Referenced by computeValues().

16.11.5.7 std::array<int, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim [protected], [inherited]

Definition at line 142 of file base_edge.h.

16.11.5.8 const int teb::BaseEdge< D, FuncType, Vertices >::Dimension [static], [inherited]

Definition at line 50 of file base_edge.h.

16.11.5.9 constexpr const int teb::BaseEdge< D, FuncType, Vertices >::NoVertices [static], [inherited]

Definition at line 48 of file base_edge.h.

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

- common_teb_edges.h

16.12 teb::EdgeQuadraticForm< p, q > Class Template Reference

Objective edge: quadratic form for states and control input.

`#include <common_teb_edges.h>`

Inheritance diagram for teb::EdgeQuadraticForm< p, q >:

Public Types

- using ValueVector = Eigen::Matrix< double, D, 1 >

  Typedef to represent the static Eigen::Vector of cost/constraint values.
- using EdgeContainer = std::vector< EdgeType ∗ >

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).
- using ValueVectorMap = Eigen::Map< Eigen::VectorXd >

  Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.

Public Member Functions

- EdgeQuadraticForm (StateVertex< p > &state_vertex, ControlVertex< q > &control_vertex)

  Construct edge and attach the corresponding state and control vertex.
- virtual void computeValues ()

  Actual cost function or constraint function. For constraints it is computeValues() <= 0.
- virtual void computeJacobian ()

  Specification of the analytic block jacobian.
- virtual void computeHessian ()

  Specification of the analytic block hessian.
- void setWeights (const Eigen::DiagonalMatrix< double, p > &Q, const Eigen::DiagonalMatrix< double, q > &R)

  Set weights $\mathbf{Q}$ *and* $\mathbf{R}$ *for the cost function.*
- void setWeights (const Eigen::Ref< const Eigen::Matrix< double, p, 1 >> &Q, const Eigen::Ref< const Eigen::Matrix< double, q, 1 >> &R)

  Set diagonal weights $\mathbf{Q}$ *and* $\mathbf{R}$ *for the cost function.*
- void setWeights (double Q, double R)

  Set weights $\mathbf{Q}$ *and* $\mathbf{R}$ *for the cost function using uniform diagonal elements.*
- void setReference (const Eigen::Ref< const Eigen::Matrix< double, p, 1 >> &xf)

  Set reference state.

- virtual int dimension () const
  Return edge dimension by function call.
- virtual FUNCT_TYPE functType () const
  Return function typed given by template parameter FuncType.
- virtual const double * valuesData () const
  Return pointer to cost/constraint values [dimension() x 1] (read-only).
- virtual double * valuesData ()
  Return pointer to cost/constraint values [dimension() x 1].
- virtual const ValueVector & values () const
  Return cost/constraint values as static Eigen::Vector (read-only).
- virtual ValueVector & values ()
  Return cost/constraint values as static Eigen::Vector.
- virtual void allocateMemory (bool skip_hessian=false)
  Allocate memory for JacobianWorkspace and HessianWorkspace and get vertices dimensions.
- virtual VertexType * getVertex (unsigned int idx)
  Access attached vertex with index `idx`.
- virtual const VertexType * getVertex (unsigned int idx) const
  Access attached vertex with index `idx` (read-only).
- virtual unsigned int noVertices () const
  Return the number of attached vertices.
- virtual void calculateVertexDimensions ()
  Return the number of attached vertices.
- virtual const std::array
  < VertexType *, NoVertices > & vertices () const
  Return vertex container of the edge.
- virtual const bool isBoundConstraint () const
  Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.
- virtual ValueVectorMap valuesMap ()
  Get cost/constraints values as Eigen type matrix.
- virtual void * getCustomData ()
  Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.
- virtual JacobianWorkspace & jacobians ()
  Access the jacobian workspace of this edge.
- virtual const JacobianWorkspace & jacobians () const
  Access the jacobian workspace of this edge (read-only).
- virtual HessianWorkspace & hessians ()
  Access the hessian workspace of this edge.
- virtual const HessianWorkspace & hessians () const
  Access the jacobian workspace of this edge (read-only).
- virtual void backupJacobian ()
  Make a backup of the current jacobian block matrix.
- virtual void restoreJacobian ()
  Restore jacobian block matrix from the backup stack.
- virtual void discardJacobianBackup ()
  Discard current jacobian backup.
- virtual JacobianWorkspace & getJacobianBackup ()

Static Public Attributes

- static constexpr const int NoVertices
- static const int Dimension

  Edge dimension as static member variable.

Protected Types

- typedef Eigen::Matrix< double, p, 1 > StateVector

  Typedef for a StateVector type (required here to avoid gcc bug within the next line)

Protected Attributes

- Eigen::DiagonalMatrix< double, p > _Q

  Weight for the state vector $\mathbf{x}_k$.
- Eigen::DiagonalMatrix< double, q > _R

  Weight for the control input $\mathbf{u}_k$.
- StateVector _ref_state = StateVector::Zero()

  Store reference state (final/goal state)
- ValueVector _values

  Actual cost/constraint value data object (initialized to zero).
- const std::array< VertexType *, NoVertices > _vertices

  Container that stores all attached vertices.
- std::array< int, NoVertices > _vertices_dim

  Store dimensions for all vertices (EdgeType::_vertices, calculateVertexDimensions())
- JacobianWorkspace _jacobians

  Block-Jacobians of the edge (see JacobianWorkspace).
- HessianWorkspace _hessians

  Block-Hessians of the edge (see HessianWorkspace)
- BackupStackType < JacobianWorkspace > _jacob_backup

16.12.1 Detailed Description

template< int p, int q> class teb::EdgeQuadraticForm< p, q >

The following cost function is implemented:

$$J_k = (\mathbf{x}_k - \mathbf{x}_{ref})^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_{ref}) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k$$

$\mathbf{Q}$ denotes the weight for the state vector $\mathbf{x}_k$. and $\mathbf{R}$ denotes the weight for the control input vector $\mathbf{u}_k$.

If this edge is added to all samples (states and control inputs) while composing the hyper-graph, it generates the following cost-function for the underlying optimization problem:

$$J = (\mathbf{x}_n - \mathbf{x}_{ref})^T \mathbf{Q}_n (\mathbf{x}_n - \mathbf{x}_{ref}) + \sum_{k=0}^{n-1} ((\mathbf{x}_k - \mathbf{x}_{ref})^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_{ref}) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k)$$

The weight for the final constraint (and for all others) can be choosen by specifying different weights. Set the weights $\mathbf{Q}$ and $\mathbf{R}$ using setWeights().

Vertices required (connect the edge with the following vertices):

1. StateVertex $\mathbf{x}_k$
2. ControlVertex $\mathbf{u}_k$

If you only want to apply the quadratic form to either only states or control inputs, set q=0 or p=0. A specialized template is provided.

Todo Currently the Levenberg-Marquardt solver take the cost function squared by itself, therefore we need to implement the sqrt here: For other solvers this cost function will be problematic without taking the square.

Remarks

We decided to allow only diagonal matrices for the weights at the moment in order to improve speed.
Call allocateMemory() before using the edge for optimization.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Definition at line 168 of file common_teb_edges.h.

16.12.2 Member Typedef Documentation

16.12.2.1 using teb::EdgeType::EdgeContainer = std::vector<EdgeType∗> [inherited]

Definition at line 114 of file graph.h.

16.12.2.2 template<int p, int q> typedef Eigen::Matrix<double,p,1> teb::EdgeQuadraticForm< p, q >::StateVector [protected]

Definition at line 265 of file common_teb_edges.h.

16.12.2.3 using teb::BaseEdge< D, FuncType, Vertices >::ValueVector = Eigen::Matrix<double, D, 1> [inherited]

Definition at line 54 of file base_edge.h.

16.12.2.4 using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd > [inherited]

Definition at line 116 of file graph.h.

16.12.3 Constructor & Destructor Documentation

16.12.3.1 template<int p, int q> teb::EdgeQuadraticForm< p, q >::EdgeQuadraticForm (StateVertex< p > & *state_vertex*, ControlVertex< q > & *control_vertex*) [inline]

Parameters

state_vertex	Reference to the StateVertex

control_vertex	Reference to the ControlVertex

Definition at line 177 of file common_teb_edges.h.

16.12.4 Member Function Documentation

16.12.4.1 virtual void teb::BaseEdge< D, FuncType, Vertices >::allocateMemory (bool *skip_hessian* = false) [inline], [virtual], [inherited]

Remarks

Before calling, call resizeVertexContainer() edge and add all vertices (setVertex()) first.

Todo Implement automatic memory allocation instead calling this function manually.

Parameters

skip_hessian	If no hessian calculation is required by the solver skip memory allocation.

Implements teb::EdgeType.

Definition at line 84 of file base_edge.h.

16.12.4.2 virtual void teb::EdgeType::backupJacobian ( ) [inline], [virtual], [inherited]

Definition at line 162 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.12.4.3 virtual void teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ( ) [inline], [virtual], [inherited]

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container EdgeType::_vertices_dim. This function is called within BaseEdge::allocate-Memory().

Todo Maybe implement static/constexpr version of collecting all dimensions since they are nown at compile-time (maybe `http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array-`

Definition at line 110 of file base_edge.h.

16.12.4.4 template<int p, int q> virtual void teb::EdgeQuadraticForm< p, q >::computeHessian ( ) [inline], [virtual]

The hessian w.r.t. a symmetric weight matrix $\mathbf{A}$ is defined by

$$\frac{\partial^2}{\partial \mathbf{x}, \partial \mathbf{x}} \mathbf{x}^T \mathbf{A} \mathbf{x} = 2\mathbf{A}$$

Applying this formular to the states and control inputs with respect to weights $\mathbf{Q}$ and $\mathbf{R}$ implies a block diagonal matrix (even a diagonal matrix, since the blocks are diagonal as well).

Reimplemented from teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >.

Definition at line 217 of file common_teb_edges.h.

References teb::EdgeType::_hessians, teb::EdgeQuadraticForm< p, q >::_Q, teb::EdgeQuadraticForm< p, q >::-_R, and teb::HessianWorkspace::getWorkspace().

16.12.4.5 template<int p, int q> virtual void teb::EdgeQuadraticForm< p, q >::computeJacobian ( ) [inline], [virtual]

It is:

$$\frac{\partial}{\partial \mathbf{x}}\mathbf{x}^T\mathbf{A}\mathbf{x} = \mathbf{x}^T\mathbf{A} + \mathbf{A}^T = 2\mathbf{x}^T\mathbf{A}$$

The last step is allowed since $\mathbf{A}$ is symmetric. This comutation is now performed for both states and control inputs w.r.t $\mathbf{Q}$ and $\mathbf{R}$.

Reimplemented from teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >.

Definition at line 200 of file common_teb_edges.h.

References teb::EdgeType::_jacobians, teb::EdgeQuadraticForm< p, q >::_Q, teb::EdgeQuadraticForm< p, q >::_R, and teb::JacobianWorkspace::getWorkspace().

16.12.4.6 template<int p, int q> virtual void teb::EdgeQuadraticForm< p, q >::computeValues ( ) [inline], [virtual]

Reimplemented from teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >.

Definition at line 182 of file common_teb_edges.h.

References teb::EdgeQuadraticForm< p, q >::_Q, teb::EdgeQuadraticForm< p, q >::_R, teb::EdgeQuadraticForm< p, q >::_ref_state, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >::_values, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >::_vertices, and teb::StateVertex< p >::states().

16.12.4.7 virtual int teb::BaseEdge< D, FuncType, Vertices >::dimension ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 51 of file base_edge.h.

16.12.4.8 virtual void teb::EdgeType::discardJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 171 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.12.4.9 virtual FUNCT_TYPE teb::BaseEdge< D, FuncType, Vertices >::functType ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 69 of file base_edge.h.

16.12.4.10 virtual void* teb::EdgeType::getCustomData ( ) [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 154 of file graph.h.

16.12.4.11 virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 173 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.12.4.12 virtual VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 98 of file base_edge.h.

16.12.4.13 virtual const VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 99 of file base_edge.h.

16.12.4.14 virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline], [virtual], [inherited]

Definition at line 159 of file graph.h.

References teb::EdgeType::_hessians.

16.12.4.15 virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline], [virtual], [inherited]

Definition at line 160 of file graph.h.

References teb::EdgeType::_hessians.

16.12.4.16 virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 124 of file graph.h.

16.12.4.17 virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline], [virtual], [inherited]

Definition at line 157 of file graph.h.

References teb::EdgeType::_jacobians.

16.12.4.18 virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline], [virtual], [inherited]

Definition at line 158 of file graph.h.

References teb::EdgeType::_jacobians.

16.12.4.19 virtual unsigned int teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 101 of file base_edge.h.

16.12.4.20 virtual void teb::EdgeType::restoreJacobian ( ) [inline], [virtual], [inherited]

Definition at line 165 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.12.4.21 template<int p, int q> void teb::EdgeQuadraticForm< p, q >::setReference (const Eigen::Ref< const Eigen::Matrix< double, p, 1 >> & *xf*) [inline]

Definition at line 246 of file common_teb_edges.h.

References teb::EdgeQuadraticForm< p, q >::_ref_state.

Referenced by teb::TebController< p, q >::buildOptimizationGraph().

16.12.4.22 template<int p, int q> void teb::EdgeQuadraticForm< p, q >::setWeights (const Eigen::DiagonalMatrix< double, p > & *Q*, const Eigen::DiagonalMatrix< double, q > & *R*) [inline]

Definition at line 225 of file common_teb_edges.h.

References teb::EdgeQuadraticForm< p, q >::_Q, and teb::EdgeQuadraticForm< p, q >::_R.

Referenced by teb::TebController< p, q >::buildOptimizationGraph().

16.12.4.23 template<int p, int q> void teb::EdgeQuadraticForm< p, q >::setWeights (const Eigen::Ref< const Eigen::Matrix< double, p, 1 >> & *Q*, const Eigen::Ref< const Eigen::Matrix< double, q, 1 >> & *R*) [inline]

Definition at line 232 of file common_teb_edges.h.

References teb::EdgeQuadraticForm< p, q >::_Q, and teb::EdgeQuadraticForm< p, q >::_R.

16.12.4.24 template<int p, int q> void teb::EdgeQuadraticForm< p, q >::setWeights (double *Q*, double *R*) [inline]

Definition at line 239 of file common_teb_edges.h.

References teb::EdgeQuadraticForm< p, q >::_Q, and teb::EdgeQuadraticForm< p, q >::_R.

16.12.4.25 virtual const ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) const [inline], [virtual], [inherited]

Definition at line 74 of file base_edge.h.

16.12.4.26 virtual ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) [inline], [virtual], [inherited]

Definition at line 75 of file base_edge.h.

16.12.4.27 virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 72 of file base_edge.h.

16.12.4.28 virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 73 of file base_edge.h.

16.12.4.29 virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline], [virtual], [inherited]

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References teb::EdgeType::dimension(), and teb::EdgeType::valuesData().

16.12.4.30 virtual const std::array<VertexType*,NoVertices>& teb::BaseEdge< D, FuncType, Vertices >::vertices ( ) const [inline], [virtual], [inherited]

Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file base_edge.h.

16.12.5 Member Data Documentation

16.12.5.1 HessianWorkspace teb::EdgeType::_hessians [protected], [inherited]

Definition at line 178 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeMinimizeTime::computeHessian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), and teb::EdgeType::hessians().

16.12.5.2 BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup [protected], [inherited]

Definition at line 180 of file graph.h.

Referenced by teb::EdgeType::backupJacobian(), teb::EdgeType::discardJacobianBackup(), teb::EdgeType::getJacobianBackup(), and teb::EdgeType::restoreJacobian().

16.12.5.3 JacobianWorkspace teb::EdgeType::_jacobians [protected], [inherited]

Definition at line 173 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeType::backupJacobian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::EdgeMinimizeTime::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::EdgeType::jacobians(), and teb::EdgeType::restoreJacobian().

16.12.5.4 template<int p, int q> Eigen::DiagonalMatrix<double,p> teb::EdgeQuadraticForm< p, q >::_Q [protected]

Definition at line 262 of file common_teb_edges.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeValues(), and teb::EdgeQuadraticForm< p, q >::setWeights().

16.12.5.5 template<int p, int q> Eigen::DiagonalMatrix<double,q> teb::EdgeQuadraticForm< p, q >::_R [protected]

Definition at line 263 of file common_teb_edges.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeValues(), and teb::EdgeQuadraticForm< p, q >::setWeights().

16.12.5.6 template<int p, int q> StateVector teb::EdgeQuadraticForm< p, q >::_ref_state = StateVector::Zero() [protected]

Definition at line 266 of file common_teb_edges.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeValues(), and teb::EdgeQuadraticForm< p, q >::setReference().

16.12.5.7 ValueVector teb::BaseEdge< D, FuncType, Vertices >::_values [protected], [inherited]

Definition at line 139 of file base_edge.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeValues().

16.12.5.8 const std::array<VertexType*, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices [protected], [inherited]

Definition at line 141 of file base_edge.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeValues().

16.12.5.9 std::array<int, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim [protected], [inherited]

Definition at line 142 of file base_edge.h.

16.12.5.10 const int teb::BaseEdge< D, FuncType, Vertices >::Dimension [static], [inherited]

Definition at line 50 of file base_edge.h.

16.12.5.11 constexpr const int teb::BaseEdge< D, FuncType, Vertices >::NoVertices [static], [inherited]

Definition at line 48 of file base_edge.h.

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

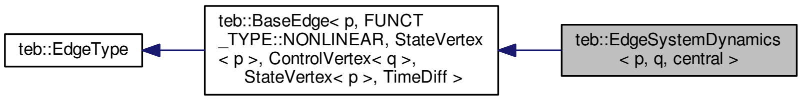
- common_teb_edges.h

16.13 teb::EdgeSystemDynamics< p, q, central > Class Template Reference

Equality constraint edge for satisfying continious system dynamics specified by an SystemDynamics object.

```
#include <base_edge_dynamics.h>
```

Inheritance diagram for teb::EdgeSystemDynamics< p, q, central >:



Public Types

- using ValueVector = Eigen::Matrix< double, D, 1 >

  Typedef to represent the static Eigen::Vector of cost/constraint values.
- using EdgeContainer = std::vector< EdgeType ∗ >

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).
- using ValueVectorMap = Eigen::Map< Eigen::VectorXd >

  Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.

Public Member Functions

- EdgeSystemDynamics (SystemDynamics< p, q > ∗system, StateVertex< p > &px1, ControlVertex< q > &pu1, StateVertex< p > &px2, TimeDiff &pdt)

  Constructor that requires a pointer to a SystemDynamics object.
- virtual void computeValues ()

  Actual cost function or constraint function. For constraints it is computeValues() <= 0.
- void setSystemDynamics (SystemDynamics< p, q > ∗system)

  Set SystemDynamics object.
- virtual int dimension () const

  Return edge dimension by function call.
- virtual FUNCT_TYPE functType () const

  Return function typed given by template parameter FuncType.
- virtual const double ∗ valuesData () const

  Return pointer to cost/constraint values [dimension() x 1] (read-only).
- virtual double ∗ valuesData ()

  Return pointer to cost/constraint values [dimension() x 1].
- virtual const ValueVector & values () const

  Return cost/constraint values as static Eigen::Vector (read-only).
- virtual ValueVector & values ()

  Return cost/constraint values as static Eigen::Vector.
- virtual void allocateMemory (bool skip_hessian=false)

Allocate memory for JacobianWorkspace and HessianWorkspace and get vertices dimensions.

- virtual VertexType * getVertex (unsigned int idx)

    Access attached vertex with index `idx`.

- virtual const VertexType * getVertex (unsigned int idx) const

    Access attached vertex with index `idx` (read-only).

- virtual unsigned int noVertices () const

    Return the number of attached vertices.

- virtual void calculateVertexDimensions ()

    Return the number of attached vertices.

- virtual const std::array
  < VertexType *, NoVertices > & vertices () const

    Return vertex container of the edge.

- virtual void computeJacobian ()

    Actual cost function or constraint function. For constraints it is computeValues() <= 0.

- virtual void computeHessian ()

    Compute Block-Hessian and store it to EdgeType::_hessians according to the description in HessianWorkspace.

- virtual const bool isBoundConstraint () const

    Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.

- virtual ValueVectorMap valuesMap ()

    Get cost/constraints values as Eigen type matrix.

- virtual void * getCustomData ()

    Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.

- virtual JacobianWorkspace & jacobians ()

    Access the jacobian workspace of this edge.

- virtual const JacobianWorkspace & jacobians () const

    Access the jacobian workspace of this edge (read-only).

- virtual HessianWorkspace & hessians ()

    Access the hessian workspace of this edge.

- virtual const HessianWorkspace & hessians () const

    Access the jacobian workspace of this edge (read-only).

- virtual void backupJacobian ()

    Make a backup of the current jacobian block matrix.

- virtual void restoreJacobian ()

    Restore jacobian block matrix from the backup stack.

- virtual void discardJacobianBackup ()

    Discard current jacobian backup.

- virtual JacobianWorkspace & getJacobianBackup ()

Static Public Attributes

- static const int Dimension = p

    Dimension of this edge equals always the number of system equations.

- static constexpr const int NoVertices

Protected Attributes

- SystemDynamics< p, q > * _system = nullptr

  Pointer to the system dynamics object.
- ValueVector _values

  Actual cost/constraint value data object (initialized to zero).
- const std::array< VertexType
  *, NoVertices > _vertices

  Container that stores all attached vertices.
- std::array< int, NoVertices > _vertices_dim

  Store dimensions for all vertices (EdgeType::_vertices, calculateVertexDimensions())
- JacobianWorkspace _jacobians

  Block-Jacobians of the edge (see JacobianWorkspace).
- HessianWorkspace _hessians

  Block-Hessians of the edge (see HessianWorkspace)
- BackupStackType
  < JacobianWorkspace > _jacob_backup

Friends

- template<int pf, int qf>
  class SystemDynamics

16.13.1 Detailed Description

template<int p, int q, bool central = false>class teb::EdgeSystemDynamics< p, q, central >

This edge creates an equility constraint based on nonlinear system-dynamics equations. System dynamics can be specified using a SystemDynamics object. The pointer to the system dynamics must be passed to the constructor of this object or by calling setSystemObject(). See SystemDynamics for more information about how to model a specific system.

This edge approximates the continuous-timde derivatives that appear in each system equation by finite differences (see teb::FiniteDifferences). Two different types of finite differences are supported by this edge at the moment.

- Forward differences: $\dot{x}(t) = \frac{x_{k+1} - x_k}{\Delta T}$
- Central differences: $\dot{x}(t) = \frac{x_{k+1} - x_{k-1}}{2\Delta T}$

The number of vertices required by this edge depend on whether central differences are enabled or not:

- Forward differences:
  1. StateVertex $\mathbf{x}_k$
  2. ControlVertex $\mathbf{u}_k$
  3. StateVertex $\mathbf{x}_{k+1}$
  4. TimeDiff ΔT
- Central differences:
  1. StateVertex $\mathbf{x}_{k-1}$
  2. StateVertex $\mathbf{x}_k$
  3. ControlVertex $\mathbf{u}_k$
  4. StateVertex $\mathbf{x}_{k+1}$

  1. TimeDiff ΔT

Remarks

Call allocateMemory() before using the edge for optimization.

Bug Central differences do not seem to work as expected at the moment.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Template Parameters

p	Number of state variables
q	Number of input variables
central	Enable central differences (make sure to read the info text -> vertex type may change)

Definition at line 52 of file base_edge_dynamics.h.

16.13.2 Member Typedef Documentation

16.13.2.1 using **teb::EdgeType::EdgeContainer** = std::vector<**EdgeType***> [inherited]

Definition at line 114 of file graph.h.

16.13.2.2 using **teb::BaseEdge**< D, FuncType, Vertices >::**ValueVector** = Eigen::Matrix<double, D, 1> [inherited]

Definition at line 54 of file base_edge.h.

16.13.2.3 using **teb::EdgeType::ValueVectorMap** = Eigen::Map< Eigen::VectorXd > [inherited]

Definition at line 116 of file graph.h.

16.13.3 Constructor & Destructor Documentation

16.13.3.1 template<int p, int q, bool central = false> **teb::EdgeSystemDynamics**< p, q, central >::**EdgeSystemDynamics** (**SystemDynamics**< p, q > * *system*, **StateVertex**< p > & *px1*, **ControlVertex**< q > & *pu1*, **StateVertex**< p > & *px2*, **TimeDiff** & *pdt*) [inline]

Parameters

system	Pointer to a SystemDynamics object
px1	Pointer to a StateVertex related to $\mathbf{x}_{k-1}$
pu1	Pointer to a ControlVertex related to $\mathbf{u}_k$
px2	Pointer to a StateVertex related to $\mathbf{x}_k$
pdt	Pointer to a TimeDiff ΔT

Definition at line 71 of file base_edge_dynamics.h.

16.13.4 Member Function Documentation

16.13.4.1 virtual void **teb::BaseEdge**< D, FuncType, Vertices >::allocateMemory (bool *skip_hessian* = false) [inline], [virtual], [inherited]

Remarks

Before calling, call resizeVertexContainer() edge and add all vertices (setVertex()) first.

Todo Implement automatic memory allocation instead calling this function manually.

Parameters

skip_hessian	If no hessian calculation is required by the solver skip memory allocation.

Implements teb::EdgeType.

Definition at line 84 of file base_edge.h.

16.13.4.2 virtual void teb::EdgeType::backupJacobian ( ) [inline], [virtual], [inherited]

Definition at line 162 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.13.4.3 virtual void teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ( ) [inline], [virtual], [inherited]

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container EdgeType::_vertices_dim. This function is called within BaseEdge::allocate-Memory().

Todo Maybe implement static/constexpr version of collecting all dimensions since they are nown at compile-time (maybe `http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array-`

Definition at line 110 of file base_edge.h.

16.13.4.4 virtual void teb::BaseEdge< D, FuncType, Vertices >::computeHessian ( ) [virtual], [inherited]

This implementation calculates the Block-Hessians for all statically allocated vertices and cost/constraint values numerically using forward differences.

For more information about the Hessian structure refer to HessianWorkspace.

If the FuncType template parameter is set to FUNCT_TYPE::LINEAR or FuncType == FUNCT_TYPE::NONLINEA-R_ONCE_DIFF the hessian is set to zero.

The results are stored to the class member BaseEdgeBinary::_hessians.

Feel free to reimplement this function in child classes in order to provide analytical Hessians.

Remarks

This function assumes that computeJacobian() was called before! It is used as reference for calculating the forward-differences.

See Also

computeJacobian(), HessianWorkspace

Implements teb::EdgeType.

16.13.4.5 virtual void teb::BaseEdge< D, FuncType, Vertices >::computeJacobian ( ) [virtual], [inherited]

This implementation calculates the Block-Jacobians for all statically allocated vertices numerically using central differences.

Compute Block-Jacobian and store it to EdgeType::_jacobians according to the description in JacobianWorkspace.

For more information about the Jacobian structure refer to JacobianWorkspace.

The results are stored to the class member BaseEdgeBinary::_jacobians.

The implementation is similar to [2].

Feel free to reimplement this function in child classes in order to provide analytical Jacobians.

See Also

computeHessian(), JacobianWorkspace

Implements teb::EdgeType.

16.13.4.6 template<int p, int q, bool central = false> virtual void teb::EdgeSystemDynamics< p, q, central >::computeValues ( ) [inline], [virtual]

Reimplemented from teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >.

Definition at line 76 of file base_edge_dynamics.h.

References teb::EdgeSystemDynamics< p, q, central >::_system, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >::_values, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >::_vertices, and teb::StateVertex< p >::states().

16.13.4.7 virtual int teb::BaseEdge< D, FuncType, Vertices >::dimension ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 51 of file base_edge.h.

16.13.4.8 virtual void teb::EdgeType::discardJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 171 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.13.4.9 virtual FUNCT_TYPE teb::BaseEdge< D, FuncType, Vertices >::functType ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 69 of file base_edge.h.

16.13.4.10 virtual void* teb::EdgeType::getCustomData ( ) [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 154 of file graph.h.

16.13.4.11 virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline], [virtual], [inherited]

Definition at line 173 of file graph.h.

References teb::EdgeType::_jacob_backup.

16.13.4.12 virtual VertexType∗ teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 98 of file base_edge.h.

16.13.4.13 virtual const VertexType∗ teb::BaseEdge< D, FuncType, Vertices >::getVertex (unsigned int *idx*) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 99 of file base_edge.h.

16.13.4.14 virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline], [virtual], [inherited]

Definition at line 159 of file graph.h.

References teb::EdgeType::_hessians.

16.13.4.15 virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline], [virtual], [inherited]

Definition at line 160 of file graph.h.

References teb::EdgeType::_hessians.

16.13.4.16 virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline], [virtual], [inherited]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 124 of file graph.h.

16.13.4.17 virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline], [virtual], [inherited]

Definition at line 157 of file graph.h.

References teb::EdgeType::_jacobians.

16.13.4.18 virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline], [virtual], [inherited]

Definition at line 158 of file graph.h.

References teb::EdgeType::_jacobians.

16.13.4.19 virtual unsigned int teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 101 of file base_edge.h.

16.13.4.20 virtual void teb::EdgeType::restoreJacobian ( ) [inline], [virtual], [inherited]

Definition at line 165 of file graph.h.

References teb::EdgeType::_jacob_backup, and teb::EdgeType::_jacobians.

16.13.4.21 template<int p, int q, bool central = false> void teb::EdgeSystemDynamics< p, q, central >::setSystemDynamics (SystemDynamics< p, q > * *system*) [inline]

Parameters

system	Pointer to a SystemDynamics object

Definition at line 93 of file base_edge_dynamics.h.

References teb::EdgeSystemDynamics< p, q, central >::_system.

16.13.4.22 virtual const ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) const [inline], [virtual], [inherited]

Definition at line 74 of file base_edge.h.

16.13.4.23 virtual ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) [inline], [virtual], [inherited]

Definition at line 75 of file base_edge.h.

16.13.4.24 virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 72 of file base_edge.h.

16.13.4.25 virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline], [virtual], [inherited]

Implements teb::EdgeType.

Definition at line 73 of file base_edge.h.

16.13.4.26 virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline], [virtual], [inherited]

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References teb::EdgeType::dimension(), and teb::EdgeType::valuesData().

16.13.4.27 virtual const std::array< VertexType∗,NoVertices >& teb::BaseEdge< D, FuncType, Vertices >::vertices ( ) const [inline], [virtual], [inherited]

Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file base_edge.h.

16.13.5 Friends And Related Function Documentation

16.13.5.1 template<int p, int q, bool central = false> template<int pf, int qf> friend class SystemDynamics [friend]

Definition at line 55 of file base_edge_dynamics.h.

16.13.6 Member Data Documentation

16.13.6.1 HessianWorkspace teb::EdgeType::_hessians [protected], [inherited]

Definition at line 178 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeMinimizeTime::computeHessian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), and teb::EdgeType::hessians().

16.13.6.2 BackupStackType< JacobianWorkspace > teb::EdgeType::_jacob_backup [protected], [inherited]

Definition at line 180 of file graph.h.

Referenced by teb::EdgeType::backupJacobian(), teb::EdgeType::discardJacobianBackup(), teb::EdgeType::getJacobianBackup(), and teb::EdgeType::restoreJacobian().

16.13.6.3 JacobianWorkspace teb::EdgeType::_jacobians [protected], [inherited]

Definition at line 173 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeType::backupJacobian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::EdgeMinimizeTime::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::EdgeType::jacobians(), and teb::EdgeType::restoreJacobian().

16.13.6.4 template<int p, int q, bool central = false> SystemDynamics<p, q>∗ teb::EdgeSystemDynamics< p, q, central >::_system = nullptr [protected]

Definition at line 102 of file base_edge_dynamics.h.

Referenced by teb::EdgeSystemDynamics< p, q, central >::computeValues(), and teb::EdgeSystemDynamics< p, q, central >::setSystemDynamics().

16.13.6.5 ValueVector teb::BaseEdge< D, FuncType, Vertices >::_values [protected], [inherited]

Definition at line 139 of file base_edge.h.

Referenced by teb::EdgeSystemDynamics< p, q, central >::computeValues().

16.13.6.6 const std::array<VertexType∗, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices [protected], [inherited]

Definition at line 141 of file base_edge.h.

Referenced by teb::EdgeSystemDynamics< p, q, central >::computeValues().

16.13.6.7 std::array<int, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim [protected], [inherited]

Definition at line 142 of file base_edge.h.

16.13.6.8 template<int p, int q, bool central = false> const int teb::EdgeSystemDynamics< p, q, central >::Dimension = p [static]

Definition at line 59 of file base_edge_dynamics.h.

16.13.6.9 constexpr const int teb::BaseEdge< D, FuncType, Vertices >::NoVertices [static], [inherited]

Definition at line 48 of file base_edge.h.

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

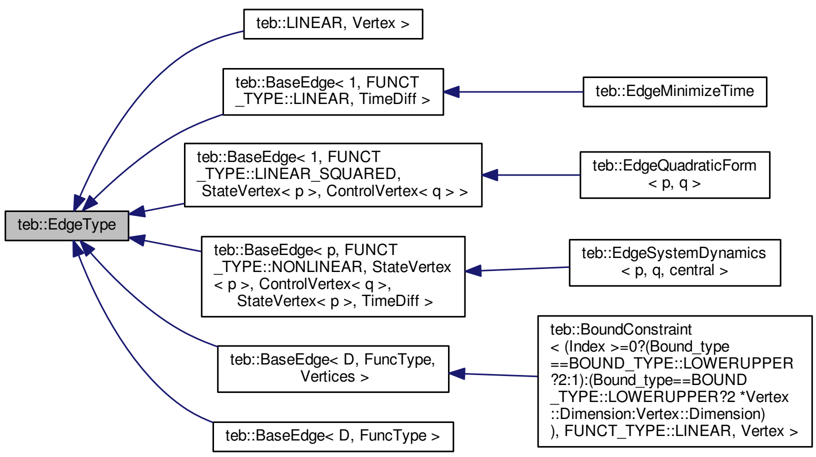
- base_edge_dynamics.h

16.14 teb::EdgeType Class Reference

Generic interface class for edges.

```
#include <graph.h>
```

Inheritance diagram for teb::EdgeType:



Public Types

- using EdgeContainer = std::vector< EdgeType ∗ >

Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).
- using ValueVectorMap = Eigen::Map< Eigen::VectorXd >

  Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.

Public Member Functions

- virtual FUNCT_TYPE functType () const =0

  Store edge function type information according to the members of enum FUNCT_TYPE.
- virtual const bool isBoundConstraint () const

  Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.
- EdgeType ()

  Constructor that updates the internal edge_type.
- virtual ∼EdgeType ()
- virtual int dimension () const =0

  Empty destructor.
- virtual void allocateMemory (bool skip_hessian=false)=0

  Allocate memory to store cost/constraint values, jacobians and hessians.
- virtual void computeValues ()=0

  Actual cost function or constraint function. For constraints it is computeValues() <= 0.
- virtual void computeJacobian ()=0

  Compute Block-Jacobian and store it to EdgeType::_jacobians according to the description in JacobianWorkspace.
- virtual void computeHessian ()=0

  Compute Block-Hessian and store it to EdgeType::_hessians according to the description in HessianWorkspace.
- virtual const double ∗ valuesData () const =0

  Return pointer to cost/constraint values [dimension() x 1] (read-only).
- virtual double ∗ valuesData ()=0

  Return pointer to cost/constraint values [dimension() x 1].
- virtual unsigned int noVertices () const =0

  Return the number of attached vertices.
- virtual VertexType ∗ getVertex (unsigned int idx)=0

  Access attached vertex with index idx.
- virtual const VertexType ∗ getVertex (unsigned int idx) const =0

  Access attached vertex with index idx (read-only).
- virtual ValueVectorMap valuesMap ()

  Get cost/constraints values as Eigen type matrix.
- virtual void ∗ getCustomData ()

  Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.
- virtual JacobianWorkspace & jacobians ()

  Access the jacobian workspace of this edge.
- virtual const JacobianWorkspace & jacobians () const

  Access the jacobian workspace of this edge (read-only).
- virtual HessianWorkspace & hessians ()

  Access the hessian workspace of this edge.
- virtual const HessianWorkspace & hessians () const

  Access the jacobian workspace of this edge (read-only).
- virtual void backupJacobian ()

  Make a backup of the current jacobian block matrix.
- virtual void restoreJacobian ()

  Restore jacobian block matrix from the backup stack.
- virtual void discardJacobianBackup ()

  Discard current jacobian backup.
- virtual JacobianWorkspace & getJacobianBackup ()

Protected Attributes

- JacobianWorkspace _jacobians

  Block-Jacobians of the edge (see JacobianWorkspace).
- HessianWorkspace _hessians

  Block-Hessians of the edge (see HessianWorkspace)
- BackupStackType
  $<$ JacobianWorkspace $>$ _jacob_backup

16.14.1 Detailed Description

This abstract class defines the interface for dedicated edges. The underlying TEB optimization problem is formulated as a hyper-graph in which state vectors (StateVertex), control input vectors (ControlVertex) and a TimeDiff are represented as vertices. Cost functions and constraints (that depend on vertices) are formulated as multi-edges. Multi-edges can connect an arbitary number of vertices. The result graph is therefore called hyper-graph. If the number of vertices attached to a single edge is low, the resulting optimization problem will be sparse probably. Block-Jacobians and Block-Hessians can be calculated for each edge independently and may be combined afterwards by the solver.

Author

Christoph Rösmann (`christoph.roesmann@tu-dortmund.de`)

See Also

HyperGraph, VertexType, BaseEdge

Definition at line 109 of file graph.h.

16.14.2 Member Typedef Documentation

16.14.2.1 using teb::EdgeType::EdgeContainer = std::vector$<$EdgeType$*$$>$

Definition at line 114 of file graph.h.

16.14.2.2 using teb::EdgeType::ValueVectorMap = Eigen::Map$<$ Eigen::VectorXd $>$

Definition at line 116 of file graph.h.

16.14.3 Constructor & Destructor Documentation

16.14.3.1 teb::EdgeType::EdgeType ( ) `[inline]`

Definition at line 127 of file graph.h.

16.14.3.2 virtual teb::EdgeType::∼EdgeType ( ) `[inline]`, `[virtual]`

Definition at line 131 of file graph.h.

16.14.4 Member Function Documentation

16.14.4.1 virtual void teb::EdgeType::allocateMemory (bool *skip_hessian* = false) [pure virtual]

Implemented in teb::BaseEdge< D, FuncType >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, and teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

16.14.4.2 virtual void teb::EdgeType::backupJacobian ( ) [inline], [virtual]

Definition at line 162 of file graph.h.

References _jacob_backup, and _jacobians.

16.14.4.3 virtual void teb::EdgeType::computeHessian ( ) [pure virtual]

Implemented in teb::BaseEdge< D, FuncType >, teb::EdgeQuadraticForm< p, q >, teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >, and teb::EdgeMinimizeTime.

16.14.4.4 virtual void teb::EdgeType::computeJacobian ( ) [pure virtual]

Implemented in teb::BaseEdge< D, FuncType >, teb::EdgeQuadraticForm< p, q >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >, teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >, and teb::EdgeMinimizeTime.

16.14.4.5 virtual void teb::EdgeType::computeValues ( ) [pure virtual]

Implemented in teb::EdgeQuadraticForm< p, q >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >, teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >, teb::EdgeSystemDynamics< p, q, central >, and teb::EdgeMinimizeTime.

16.14.4.6 virtual int teb::EdgeType::dimension ( ) const [pure virtual]

Return number of scalar cost/constraint values covered by this edge.

Implemented in teb::BaseEdge< D, FuncType >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, and teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

Referenced by valuesMap().

16.14.4.7 virtual void teb::EdgeType::discardJacobianBackup ( ) [inline], [virtual]

Definition at line 171 of file graph.h.

References _jacob_backup.

16.14.4.8 virtual FUNCT_TYPE teb::EdgeType::functType ( ) const [pure virtual]

Implemented in teb::BaseEdge< D, FuncType >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, and teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

16.14.4.9 virtual void* teb::EdgeType::getCustomData ( ) [inline], [virtual]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 154 of file graph.h.

16.14.4.10 virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline], [virtual]

Definition at line 173 of file graph.h.

References _jacob_backup.

16.14.4.11 virtual VertexType* teb::EdgeType::getVertex (unsigned int *idx*) [pure virtual]

Implemented in teb::BaseEdge< D, FuncType >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, and teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

16.14.4.12 virtual const VertexType* teb::EdgeType::getVertex (unsigned int *idx*) const [pure virtual]

Implemented in teb::BaseEdge< D, FuncType >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, and teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

16.14.4.13 virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline], [virtual]

Definition at line 159 of file graph.h.

References _hessians.

16.14.4.14 virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline], [virtual]

Definition at line 160 of file graph.h.

References _hessians.

16.14.4.15 virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline], [virtual]

Reimplemented in teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >.

Definition at line 124 of file graph.h.

16.14.4.16 virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline], [virtual]

Definition at line 157 of file graph.h.

References _jacobians.

16.14.4.17 virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline], [virtual]

Definition at line 158 of file graph.h.

References _jacobians.

16.14.4.18 virtual unsigned int teb::EdgeType::noVertices ( ) const [pure virtual]

Implemented in teb::BaseEdge< D, FuncType >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, and teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

16.14.4.19 virtual void teb::EdgeType::restoreJacobian ( ) [inline], [virtual]

Definition at line 165 of file graph.h.

References _jacob_backup, and _jacobians.

16.14.4.20 virtual const double* teb::EdgeType::valuesData ( ) const [pure virtual]

Implemented in teb::BaseEdge< D, FuncType >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, and teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

Referenced by valuesMap().

16.14.4.21 virtual double* teb::EdgeType::valuesData ( ) [pure virtual]

Implemented in teb::BaseEdge< D, FuncType >, teb::BaseEdge< D, FuncType, Vertices >, teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >, teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >, teb::LINEAR, Vertex >, and teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >.

16.14.4.22 virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline], [virtual]

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References dimension(), and valuesData().

16.14.5 Member Data Documentation

16.14.5.1 HessianWorkspace teb::EdgeType::_hessians [protected]

Definition at line 178 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeMinimizeTime::computeHessian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), and hessians().

16.14.5.2 BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup [protected]

Definition at line 180 of file graph.h.

Referenced by backupJacobian(), discardJacobianBackup(), getJacobianBackup(), and restoreJacobian().

16.14.5.3 JacobianWorkspace teb::EdgeType::_jacobians [protected]

Definition at line 173 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), backupJacobian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::EdgeMinimizeTime::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::BaseEdge< D, FuncType >::computeJacobian(), jacobians(), and restoreJacobian().

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

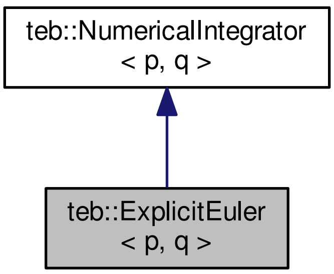
- graph.h

16.15 teb::ExplicitEuler< p, q > Class Template Reference

Simple explicit euler method for integration.

`#include <integrators.h>`

Inheritance diagram for teb::ExplicitEuler< p, q >:



Public Types

- using StateVector = typename NumericalIntegrator< p, q >::StateVector

  Typedef for state vector with p states [p x 1].
- using ControlVector = typename NumericalIntegrator< p, q >::ControlVector

  Typedef for control input vector with q controls [q x 1].

Public Member Functions

- ExplicitEuler ()

  Empty Constructor.

- virtual ~ExplicitEuler ()

  Empty Destructor.

- virtual StateVector integrate (const Eigen::Ref< const StateVector > &x_k, const Eigen::Ref< const ControlVector > &u_k, SystemDynamics< p, q > *system_equation, double dt) const

  Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

  .

16.15.1 Detailed Description

template<int p, int q>class teb::ExplicitEuler< p, q >

For more information please refer to http://en.wikipedia.org/wiki/Euler_method

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Template Parameters

p	Number of state variables
q	Number of input variables

Definition at line 81 of file integrators.h.

16.15.2 Member Typedef Documentation

16.15.2.1 template<int p, int q> using teb::ExplicitEuler< p, q >::ControlVector = typename NumericalIntegrator<p,q>::ControlVector

Definition at line 87 of file integrators.h.

16.15.2.2 template<int p, int q> using teb::ExplicitEuler< p, q >::StateVector = typename NumericalIntegrator<p,q>::StateVector

Definition at line 85 of file integrators.h.

16.15.3 Constructor & Destructor Documentation

16.15.3.1 template<int p, int q> teb::ExplicitEuler< p, q >::ExplicitEuler ( ) [inline]

Definition at line 89 of file integrators.h.

16.15.3.2 template<int p, int q> virtual teb::ExplicitEuler< p, q >::~ExplicitEuler ( ) [inline], [virtual]

Definition at line 90 of file integrators.h.

16.15.4 Member Function Documentation

16.15.4.1 template< int p, int q > virtual **StateVector** teb::ExplicitEuler< p, q >::integrate (const Eigen::Ref< const **StateVector** > & *x_k*, const Eigen::Ref< const **ControlVector** > & *u_k*, **SystemDynamics**< p, q > * *system_equation*, double *dt*) const [inline], [virtual]

Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

w.r.t. $\mathbf{x}$. In particular, it relies on discrete states. The method calculates $\mathbf{x}_{k+1}$ based on $\mathbf{x}_k$, $\mathbf{u}_k$ and the duration ΔT. The dedicated system dynamics equations may be specified using a SystemDynamics object.

Parameters

x_k	Current state vector $\mathbf{x}_k$ [p x 1]
u_k	Current state vector $\mathbf{u}_k$ [q x 1]
system_equation	Pointer to the SystemDynamics object that stores the system equations.
dt	Time interval for the integration

Returns

Integrated state vector $\mathbf{x}_{k+1}$ [p x 1]

Implements teb::NumericalIntegrator< p, q >.

Definition at line 93 of file integrators.h.

References teb::SystemDynamics< p, q >::stateSpaceModel().

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

- integrators.h

16.16 teb::Config::Optim::Solver::NonlinearProgram::Hessian Struct Reference

Configurations for the hessian of the lagrangian calculation.

```
#include <config.h>
```

Public Attributes

- HessianMethod hessian_method = HessianMethod::NUMERIC

  Define the hessian calcuation methods (numeric, bfgs quasi-newton, ...)
- HessianInit hessian_init = HessianInit::NUMERIC

  Define which hessian initialization should be used in BFGS methods.
- double hessian_init_identity_scale = 1

  Scale the hessian initialization matrix if hessian_init is set to HessianInit::IDENTITY.
- bool bfgs_damped_mode = true

  Specify if BFGS damped mode should be used (slower, but more robust).

16.16.1 Detailed Description

Definition at line 96 of file config.h.

16.16.2 Member Data Documentation

16.16.2.1 bool teb::Config::Optim::Solver::NonlinearProgram::Hessian::bfgs_damped_mode = true

Definition at line 101 of file config.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS().

16.16.2.2 HessianInit teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init = HessianInit::NUMERIC

Definition at line 99 of file config.h.

Referenced by teb::BaseSolverNonlinearProgramDense::initHessianBFGS().

16.16.2.3 double teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init_identity_scale = 1

Definition at line 100 of file config.h.

Referenced by teb::BaseSolverNonlinearProgramDense::initHessianBFGS().

16.16.2.4 HessianMethod teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_method = HessianMethod::NUMERIC

Definition at line 98 of file config.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::SolverSQPDense::initSolverWorkspace(), and teb::SolverSQPDense::solveImpl().

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

- config.h

16.17 teb::HessianWorkspace Class Reference

Memory management and accessor functions for the Block-Hessian (for each edge).

```
#include <workspaces.h>
```

Public Types

- using WorkspaceMatrix = std::vector< std::vector< Eigen::MatrixXd, Eigen::aligned_allocator< Eigen::MatrixXd >>>

  Typedef for the BlockHessian container.

Public Member Functions

- HessianWorkspace ()
- ∼HessianWorkspace ()

  Empty constructor.
- template<typename T >
  bool allocate (int edge_dim, const T &vertices_dim)

  Empty destructor.
- Eigen::MatrixXd & getWorkspace (int vertex_idx, int value_idx)

  Access Block-Hessian for TebVertex with index `vertex_idx`.

Protected Attributes

- WorkspaceMatrix _workspace

  Block-Hessian container.

16.17.1 Detailed Description

This class controls the memory management of Block-Hessians that are calculated for each edge (depends on the BaseSolver subclass, if Hessians are required). Each edge instantiates its own HessianWorkspace and allocates the memory required to store all entries. Since the size depends on the type and number of attached vertices, it is not known a-priori. This workspace helps managing memory and accessing elements at run-time.

Remarks

For each vertex attached to a subclass of BaseEdge and addition for each cost-value (!), a single Hessian of the size [TebVertex::dimension() x TebVertex::dimension()] is required.
Assume that there exist a vertex1 with the following unfixed variables: [x1, x2, x3]. In addition let vertex2 be a vertex with the unfixed variables: [x4,x5]. For the an edge with two cost/constraint values [v1, v2], the corresponding Block-Hessians are as follow:

$$\mathbf{H}_{block1,vertex1} = \begin{bmatrix} \partial^2_{x1,x1}\ v1 & \partial^2_{x1,x2}\ v1 & \partial^2_{x1,x3}\ v1 \\ \partial^2_{x2,x1}\ v1 & \partial^2_{x2,x2}\ v1 & \partial^2_{x2,x3}\ v1 \\ \partial^2_{x3,x1}\ v1 & \partial^2_{x3,x2}\ v1 & \partial^2_{x3,x3}\ v1 \end{bmatrix}$$

$$\mathbf{H}_{block1,vertex2} = \begin{bmatrix} \partial^2_{x4,x4}\ v1 & \partial^2_{x4,x5}\ v1 \\ \partial^2_{x5,x4}\ v1 & \partial^2_{x5,x5}\ v1 \end{bmatrix}$$

$$\mathbf{H}_{block2,vertex1} = \begin{bmatrix} \partial^2_{x1,x1}\ v2 & \partial^2_{x1,x2}\ v2 & \partial^2_{x1,x3}\ v2 \\ \partial^2_{x2,x1}\ v2 & \partial^2_{x2,x2}\ v2 & \partial^2_{x2,x3}\ v2 \\ \partial^2_{x3,x1}\ v2 & \partial^2_{x3,x2}\ v2 & \partial^2_{x3,x3}\ v2 \end{bmatrix}$$

$$\mathbf{H}_{block2,vertex2} = \begin{bmatrix} \partial^2_{x4,x4}\ v2 & \partial^2_{x4,x5}\ v2 \\ \partial^2_{x5,x4}\ v2 & \partial^2_{x5,x5}\ v2 \end{bmatrix}$$

The above Block-Hessians can be stored from the edge class (after calling allocate() once) as follows:

```
this->getWorkspace(0,0) = H_block1_vertex1; // exchange "this" by a pointer/ref to a valid
      workspace instance with allocated and reserved memory.
this->getWorkspace(1,0) = H_block1_vertex2;
this->getWorkspace(1,1) = H_block2_vertex1;
this->getWorkspace(1,1) = H_block2_vertex2;
```

Todo TebVertex::dimension() is used at the moment. Acutally only a number of TebVertex::dimensionFree() non-zeros may appear. Test if it speeds up the optimization to skip those values by using the fixed-mask. But for common MPC problems only the first or/and the final state are fixed partially.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

BaseEdge, JacobianWorkspace

Definition at line 153 of file workspaces.h.

16.17.2 Member Typedef Documentation

16.17.2.1 using **teb::HessianWorkspace::WorkspaceMatrix** = std::vector<std::vector<Eigen::MatrixXd, Eigen::aligned_allocator<Eigen::MatrixXd>>>

A single block-matrix per attached TebVertex and per Cost-value is required.

Definition at line 157 of file workspaces.h.

16.17.3 Constructor & Destructor Documentation

16.17.3.1 teb::HessianWorkspace::HessianWorkspace ( ) `[inline]`

Definition at line 159 of file workspaces.h.

16.17.3.2 teb::HessianWorkspace::~HessianWorkspace ( ) `[inline]`

Definition at line 160 of file workspaces.h.

16.17.4 Member Function Documentation

16.17.4.1 template<typename T > bool teb::HessianWorkspace::allocate (int *edge_dim*, const T & *vertices_dim*) `[inline]`

Allocate memory for all block-Jacobians.

Parameters

edge_dim	Dimension of the edge. Typically BaseEdge::Dimension().
vertices_dim	Vector of dimensions of the attached vertices. Typically TebVertex::dimension() each.

Template Parameters

T	storage class (std::vector, std::array, std::deque)

Return values

true	Status flag. Currently it returns only `true`.

Definition at line 170 of file workspaces.h.

References _workspace.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), and teb::BaseEdge< D, FuncType >::allocateMemory().

16.17.4.2 Eigen::MatrixXd& teb::HessianWorkspace::getWorkspace (int *vertex_idx*, int *value_idx*) `[inline]`

Parameters

vertex_idx	Index of the corresponding vertex.
value_idx	Index of the corresponding cost-value.

Returns

Reference to Block-Jacobian of vertex with index `vertex_idx` and cost value with index `value_idx`.

Definition at line 188 of file workspaces.h.

References _workspace.

Referenced by teb::EdgeMinimizeTime::computeHessian(), teb::BoundConstraint< Bound_type, Vertex, Bound_-vars, Index >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeHessian(), and teb::BaseEdge< D, FuncType >::computeHessian().

16.17.5 Member Data Documentation

16.17.5.1 WorkspaceMatrix teb::HessianWorkspace::_workspace [protected]

Definition at line 188 of file workspaces.h.

Referenced by allocate(), and getWorkspace().

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

- workspaces.h

16.18 teb::HyperGraph Class Reference

Graph object that stores pointers to active vertices and edges.

`#include <graph.h>`

Public Types

- using EdgeContainer = EdgeType::EdgeContainer
- using VertexContainer = VertexType::VertexContainer

Public Member Functions

- EdgeContainer & objectives ()

    Access objective container.
- const EdgeContainer & objectives () const

    Access all edges containing objectives (read-only)
- EdgeContainer & equalities ()

    Access equality constraint container.
- const EdgeContainer & equalities () const

    Access all edges containing inequality constraints (read-only)
- EdgeContainer & inequalities ()

    Access inequality constraint container.
- const EdgeContainer & inequalities () const

    Access all edges containing equality constraints (read-only)
- VertexContainer & activeVertices ()

    Access container that stores only the active vertices of the hyper-graph.
- const VertexContainer & activeVertices () const

    *Access all **active** vertices (read-only)*
- void clearEdges ()

    Clears all existing edges and frees memory.
- void clearActiveVertices ()

    Clears all active vertices (the actual vertices remain untouched).
- void clearGraph ()

    Clear the complete graph.

- void addActiveVertex (VertexType ∗vertex)

    Add new vertex to the graph.
- void addEdgeObjective (EdgeType ∗edge)

    Add new objective edge to the graph.
- void addEdgeInequality (EdgeType ∗edge)

    Add new inequality constraint edge to the graph.
- void addEdgeEquality (EdgeType ∗edge)

    Add new equality constraint edge to the graph.
- void notifyGraphModified (bool modified=true)

    Set status of the graph to modified.
- bool isGraphModified () const

    Return status that tracks if the graph has been modified The status is tracked in order to allow the controller class and the solver to hotstart from previous settings.

Protected Attributes

- EdgeContainer _objectives

    Store all edges containing local objective/cost functions (see buildOptimizationGraph()).
- EdgeContainer _constraints_eq

    Store all edges containing equality constraints (see buildOptimizationGraph()).
- EdgeContainer _constraints_ineq

    Store all edges containing inequality constraints (see buildOptimizationGraph()).
- VertexContainer _active_vertices

    Store all active vertices (active = at least one unfixed variable, see getActiveVertices()).
- bool _graph_modified = true

    Store status flag if the graph structure is changed or not.

16.18.1 Detailed Description

This class stores pointer to all active vertices and edges in the optimizable hyper-graph. An instance of this class needs to be created inside the controller class and afterwards it is passed to a dedicated solver class via Base-Solver::solve(). The solver solves the optimization problem based on the hyper-graph specified within this class.

Note: Active vertices are all vertices with at least one degree of freedom.

Remarks

The graph takes ownership for the memory management of the edges. Vertices remain untouched, since they are pointing to real states in the controller class (and only to the active ones, only the vector of pointers will be cleared).

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

VertexType, EdgeType, StateVertex, ControlVertex, TimeDiff, BaseEdge

Definition at line 205 of file graph.h.

16.18.2 Member Typedef Documentation

16.18.2.1 using teb::HyperGraph::EdgeContainer = EdgeType::EdgeContainer

Definition at line 208 of file graph.h.

16.18.2.2 using teb::HyperGraph::VertexContainer = VertexType::VertexContainer

Definition at line 209 of file graph.h.

16.18.3 Member Function Documentation

16.18.3.1 VertexContainer& teb::HyperGraph::activeVertices ( ) [inline]

Active vertices are vertices that store at least one free optimization variable. That means, that iff a vertex (e.g. StateVertex, ControlVertex or TimeDiff) is completly fixed, than it is not active. In that case the vertex could be skipped during optimization to speed up optimization time.

Returns

Reference to the active vertices container.

Definition at line 264 of file graph.h.

References _active_vertices.

Referenced by teb::BaseSolver::solve().

16.18.3.2 const VertexContainer& teb::HyperGraph::activeVertices ( ) const [inline]

Definition at line 266 of file graph.h.

References _active_vertices.

16.18.3.3 void teb::HyperGraph::addActiveVertex (VertexType * *vertex*) [inline]

The graph does not take over memory management for vertices.

Parameters

vertex	Pointer to the VertexType subclass.

Definition at line 312 of file graph.h.

References _active_vertices.

16.18.3.4 void teb::HyperGraph::addEdgeEquality (EdgeType * *edge*) [inline]

The graph takes care about memory management for this edge.

Parameters

edge	Pointer to the EdgeType subclass.

Definition at line 344 of file graph.h.

References _constraints_eq.

16.18.3.5 void teb::HyperGraph::addEdgeInequality (EdgeType * edge) [inline]

The graph takes care about memory management for this edge.

Parameters

edge	Pointer to the EdgeType subclass.

Definition at line 333 of file graph.h.

References _constraints_ineq.

16.18.3.6 void teb::HyperGraph::addEdgeObjective (EdgeType * *edge*) [inline]

The graph takes care about memory management for this edge.

Parameters

edge	Pointer to the EdgeType subclass.

Definition at line 322 of file graph.h.

References _objectives.

16.18.3.7 void teb::HyperGraph::clearActiveVertices ( ) [inline]

Definition at line 288 of file graph.h.

References _active_vertices, and _graph_modified.

Referenced by clearGraph().

16.18.3.8 void teb::HyperGraph::clearEdges ( ) [inline]

Clears all edges (objectives, inequality constraints and equality constraints). All memory will be freed.

Definition at line 274 of file graph.h.

References _constraints_eq, _constraints_ineq, _graph_modified, and _objectives.

Referenced by clearGraph().

16.18.3.9 void teb::HyperGraph::clearGraph ( ) [inline]

Clears all edges (objectives, inequality constraints and equality constraints). All memory for the edges will be freed. The actual vertices remain untouched.

Definition at line 300 of file graph.h.

References clearActiveVertices(), and clearEdges().

Referenced by teb::TebController< p, q >::resetController(), and teb::TebController< p, q >::~TebController().

16.18.3.10 EdgeContainer& teb::HyperGraph::equalities ( ) [inline]

Equality constraints (subclasses of BaseEdge) are inserted within TebController::buildOptimizationGraph() and TebController::customOptimizationGraph().

Returns

Reference to the equality constraint container.

Definition at line 240 of file graph.h.

References _constraints_eq.

Referenced by teb::BaseSolver::solve().

16.18.3.11 const EdgeContainer& teb::HyperGraph::equalities ( ) const [inline]

Definition at line 242 of file graph.h.

References _constraints_eq.

16.18.3.12 EdgeContainer& teb::HyperGraph::inequalities ( ) [inline]

Inequality constraints (subclasses of BaseEdge) are inserted within TebController::buildOptimizationGraph() and TebController::customOptimizationGraph().

Returns

Reference to the inequality constraint container.

Definition at line 251 of file graph.h.

References _constraints_ineq.

Referenced by teb::BaseSolver::solve().

16.18.3.13 const EdgeContainer& teb::HyperGraph::inequalities ( ) const [inline]

Definition at line 253 of file graph.h.

References _constraints_ineq.

16.18.3.14 bool teb::HyperGraph::isGraphModified ( ) const [inline]

If the structure of the optimization problem remains unchanged, one do not need to collect all active edges again. Therefore check the status using this function whenever vertices and edges are processed.

Returns

`true` if the graph has been modified

Definition at line 373 of file graph.h.

References _graph_modified.

Referenced by teb::BaseSolver::solve().

16.18.3.15 void teb::HyperGraph::notifyGraphModified (bool *modified* = `true`) [inline]

The status is tracked in order to allow the controller class and the solver to hotstart from previous settings. If the structure of the optimization problem remains unchanged, one do not need to collect all active edges again. Therefore call this function whenever the graph structure is changed (e.g. by adding/removing edges or vertices).

Parameters

modified	If `true` the graph is modified, otherwise reset the status.

Definition at line 359 of file graph.h.

References _graph_modified.

Referenced by teb::TebController< p, q >::activateControlBounds(), teb::TebController< p, q >::activateObjectiveQuadraticForm(), teb::TebController< p, q >::activateObjectiveTimeOptimal(), teb::TebController< p, q >::activateStateBounds(), teb::TebController< p, q >::setFixedDt(), teb::TebController< p, q >::setFixedGoal(), teb::TebController< p, q >::setFixedStart(), teb::TebController< p, q >::setGoalStatesFixedOrUnfixed(), and teb::TebController< p, q >::setSystemDynamics().

16.18.3.16 EdgeContainer& teb::HyperGraph::objectives ( ) [inline]

Objectives (subclasses of BaseEdge) are inserted within TebController::buildOptimizationGraph() and TebController::customOptimizationGraph() for instance.

Returns

Reference to the objective container.

Definition at line 229 of file graph.h.

References _objectives.

Referenced by teb::BaseSolver::solve().

16.18.3.17 const EdgeContainer& teb::HyperGraph::objectives ( ) const [inline]

Definition at line 231 of file graph.h.

References _objectives.

16.18.4 Member Data Documentation

16.18.4.1 VertexContainer teb::HyperGraph::_active_vertices [protected]

Definition at line 216 of file graph.h.

Referenced by activeVertices(), addActiveVertex(), and clearActiveVertices().

16.18.4.2 EdgeContainer teb::HyperGraph::_constraints_eq [protected]

Definition at line 214 of file graph.h.

Referenced by addEdgeEquality(), clearEdges(), and equalities().

16.18.4.3 EdgeContainer teb::HyperGraph::_constraints_ineq [protected]

Definition at line 215 of file graph.h.

Referenced by addEdgeInequality(), clearEdges(), and inequalities().

16.18.4.4 bool teb::HyperGraph::_graph_modified = true [protected]

Definition at line 218 of file graph.h.

Referenced by clearActiveVertices(), clearEdges(), isGraphModified(), and notifyGraphModified().

16.18.4.5 EdgeContainer teb::HyperGraph::_objectives [protected]

Definition at line 213 of file graph.h.

Referenced by addEdgeObjective(), clearEdges(), and objectives().

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

- graph.h

16.19 teb::JacobianWorkspace Class Reference

Memory management and accessor functions for the Block-Jacobians (for each edge).

`#include <workspaces.h>`

Public Types

- using WorkspaceMatrix = std::vector< Eigen::MatrixXd, Eigen::aligned_allocator< Eigen::MatrixXd >>

    Typedef for the BlockJacobian container.

Public Member Functions

- JacobianWorkspace ()
- ∼JacobianWorkspace ()

    Empty constructor.

- template< typename T >
  bool allocate (int edge_dim, const T &vertices_dim)

    Empty destructor.

- Eigen::MatrixXd & getWorkspace (int vertex_idx)

    Access Block-Jacobian for TebVertex with index `vertex_idx`.

Protected Attributes

- WorkspaceMatrix _workspace

    Block-Jacobian container.

16.19.1 Detailed Description

This class controls the memory management of Block-Jacobians that are calculated for each edge. Each edge instantiates its own JacobianWorkspace and allocates the memory required to store all entries. Since the size depends on the type and number of attached vertices and the error dimension, it is not known a-priori. This workspace helps managing memory and accessing elements at run-time.

Remarks

For each vertex attached to a subclass of BaseEdge, a single Jacobian of the size [BaseEdge::Dimension x TebVertex::dimension()] is required.
Assume that a vertex has the following unfixed variables: [x1, x2, x3] and an edge has the cost values [v1, v2]:

$$\mathbf{J}_{block} = \begin{bmatrix} \partial_{x1}\ v1 & \partial_{x2}\ v1 & \partial_{x3}\ v1 \\ \partial_{x1}\ v2 & \partial_{x2}\ v2 & \partial_{x3}\ v2 \end{bmatrix}$$

Let the vertex be the first one of the corresponding edge. The above Block-Jacobian can be stored from an edge (after calling allocate() once) as follows:

```
this->getWorkspace(0) = J_block; // exchange "this" by a pointer/ref to a valid workspace
      instance with allocated and reserved memory.
```

Todo TebVertex::dimension() is used at the moment. Acutally only a number of TebVertex::dimensionFree() non-zeros may appear. Test if it speeds up the optimization to skip those values by using the fixed-mask. But for common MPC problems only the first or/and the final state are fixed partially.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

BaseEdge, HessianWorkspace

Definition at line 48 of file workspaces.h.

16.19.2 Member Typedef Documentation

16.19.2.1 using **teb::JacobianWorkspace::WorkspaceMatrix** = std::vector<Eigen::MatrixXd, Eigen::aligned_allocator<Eigen::MatrixXd>>

A single block-matrix per attached TebVertex is required.

Definition at line 52 of file workspaces.h.

16.19.3 Constructor & Destructor Documentation

16.19.3.1 teb::JacobianWorkspace::JacobianWorkspace ( ) [inline]

Definition at line 54 of file workspaces.h.

16.19.3.2 teb::JacobianWorkspace::~JacobianWorkspace ( ) [inline]

Definition at line 55 of file workspaces.h.

16.19.4 Member Function Documentation

16.19.4.1 template<typename T > bool teb::JacobianWorkspace::allocate (int *edge_dim,* const T & *vertices_dim*) [inline]

Allocate memory for all block-Jacobians.

Parameters

edge_dim	Dimension of the edge. Typically BaseEdge::Dimension().
vertices_dim	Vector of dimensions of the attached vertices. Typically TebVertex::dimension() each.

Template Parameters

T	storage class (std::vector, std::array, std::deque)

Return values

true	Status flag. Currently it returns only `true`.

Definition at line 65 of file workspaces.h.

References _workspace.

Referenced by teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory(), and teb::BaseEdge< D, FuncType >::allocateMemory().

16.19.4.2 Eigen::MatrixXd& teb::JacobianWorkspace::getWorkspace (int *vertex_idx*) [inline]

Parameters

vertex_idx	Index of the corresponding vertex.

Returns

Reference to Block-Jacobian of vertex with index `vertex_idx`.

Definition at line 82 of file workspaces.h.

References _workspace.

Referenced by teb::BaseEdge< D, FuncType, Vertices >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::EdgeMinimizeTime::computeJacobian(), teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), and teb::BaseEdge< D, FuncType >::computeJacobian().

16.19.5 Member Data Documentation

16.19.5.1 WorkspaceMatrix teb::JacobianWorkspace::_workspace [protected]

Definition at line 82 of file workspaces.h.

Referenced by allocate(), and getWorkspace().

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

- workspaces.h

16.20 teb::Config::Optim::Solver::NonlinearProgram::LineSearch Struct Reference

Settings for the line-search algorithm (force merit function descent)

`#include <config.h>`

Public Attributes

- double sigma = 0.5
- double beta = 0.5
- double alpha_init = 0.1

16.20.1 Detailed Description

Definition at line 105 of file config.h.

16.20.2 Member Data Documentation

16.20.2.1 double teb::Config::Optim::Solver::NonlinearProgram::LineSearch::alpha_init = 0.1

Definition at line 109 of file config.h.

Referenced by teb::SolverSQPDense::initSolverWorkspace().

16.20.2.2 double teb::Config::Optim::Solver::NonlinearProgram::LineSearch::beta = 0.5

Definition at line 108 of file config.h.

Referenced by teb::SolverSQPDense::solveImpl().

16.20.2.3 double teb::Config::Optim::Solver::NonlinearProgram::LineSearch::sigma = 0.5

Definition at line 107 of file config.h.

Referenced by teb::SolverSQPDense::solveImpl().

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

- config.h

16.21 teb::Config::Optim::Solver::Lsq Struct Reference

Configurations especially for least-squares-solvers.

```
#include <config.h>
```

Public Attributes

- double soft_constr_epsilon = 0.

    Add a bigger margin to soft-constraints.
- double weight_equalities = 2

    Soft-constraint weight for equality constraints.
- double weight_inequalities = 2

    Soft-constraint weight for inequality constraints.
- double weight_adaptation_factor = 2

    Factor for soft-constraint weight adapation (see BaseSolverLeastSquares::adaptWeights()).

16.21.1 Detailed Description

Definition at line 83 of file config.h.

16.21.2 Member Data Documentation

16.21.2.1 double teb::Config::Optim::Solver::Lsq::soft_constr_epsilon = 0.

Bug Do not use at the moment, because it is not implemented correctly

Examples:

rocket_system.cpp.

Definition at line 85 of file config.h.

Referenced by teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardt-EigenSparse::buildJacobian(), and teb::BaseSolverLeastSquares::buildValueVector().

16.21.2.2 double teb::Config::Optim::Solver::Lsq::weight_adaptation_factor = 2

Examples:

integrator_system_classic_mpc.cpp, mobile_robot_teb.cpp, and rocket_system.cpp.

Definition at line 88 of file config.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights().

16.21.2.3 double teb::Config::Optim::Solver::Lsq::weight_equalities = 2

Examples:

integrator_system_classic_mpc.cpp, linear_system_ode_ctrl_comparison.cpp, and mobile_robot_teb.cpp.

Definition at line 86 of file config.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights().

16.21.2.4 double teb::Config::Optim::Solver::Lsq::weight_inequalities = 2

Examples:

integrator_system_classic_mpc.cpp, mobile_robot_teb.cpp, and rocket_system.cpp.

Definition at line 87 of file config.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights().

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

- config.h

16.22 teb::Config::Optim::Solver::Nlopt Struct Reference

Configurations especially for the nlopt solver wrapper.

```
#include <config.h>
```

Public Attributes

- NloptAlgorithms algorithm = NloptAlgorithms::SLSQP
- double tolerance_equalities = 0.001
  Stop optimization depending of the equality constraint satisfaction tolerance.
- double tolerance_inequalities = 0.001
  Stop optimization depending of the inequality constraint satisfaction tolerance.
- double stopping_criteria_ftol_abs = 0.0001
  Stop optimization depending on the absolute value of the objective function.
- double stopping_criteria_ftol_rel = 0.0001
  Stop optimization depending on the relative change of the objective function value.
- double stopping_criteria_xtol_abs = 0.0001
  Stop optimization depending on the absolute values of the optimization vector.
- double stopping_criteria_xtol_rel = 0.0001
  Stop optimization depending on the relative change of values of the optimization vector.
- double max_optimization_time = 5
  Stop optimization after a given duration in seconds.

16.22.1 Detailed Description

Definition at line 115 of file config.h.

16.22.2 Member Data Documentation

16.22.2.1 NloptAlgorithms teb::Config::Optim::Solver::Nlopt::algorithm = NloptAlgorithms::SLSQP

Definition at line 117 of file config.h.

16.22.2.2 double teb::Config::Optim::Solver::Nlopt::max_optimization_time = 5

Definition at line 126 of file config.h.

16.22.2.3 double teb::Config::Optim::Solver::Nlopt::stopping_criteria_ftol_abs = 0.0001

Definition at line 122 of file config.h.

16.22.2.4 double teb::Config::Optim::Solver::Nlopt::stopping_criteria_ftol_rel = 0.0001

Definition at line 123 of file config.h.

16.22.2.5 double teb::Config::Optim::Solver::Nlopt::stopping_criteria_xtol_abs = 0.0001

Definition at line 124 of file config.h.

16.22.2.6 double teb::Config::Optim::Solver::Nlopt::stopping_criteria_xtol_rel = 0.0001

Definition at line 125 of file config.h.

16.22.2.7 double teb::Config::Optim::Solver::Nlopt::tolerance_equalities = 0.001

Definition at line 119 of file config.h.

16.22.2.8 double teb::Config::Optim::Solver::Nlopt::tolerance_inequalities = 0.001

Definition at line 120 of file config.h.

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

- config.h

16.23 teb::Config::Optim::Solver::NonlinearProgram Struct Reference

Settings for constrained optimiziation solver (nonlinear program solver)

```
#include <config.h>
```

Classes

- struct Hessian

  Configurations for the hessian of the lagrangian calculation.
- struct LineSearch

  Settings for the line-search algorithm (force merit function descent)

Public Attributes

- struct
  teb::Config::Optim::Solver::NonlinearProgram::Hessian hessian

  Configurations for the hessian of the lagrangian calculation.
- struct
  teb::Config::Optim::Solver::NonlinearProgram::LineSearch linesearch

  Settings for the line-search algorithm (force merit function descent)

16.23.1 Detailed Description

Definition at line 93 of file config.h.

16.23.2 Member Data Documentation

16.23.2.1 struct teb::Config::Optim::Solver::NonlinearProgram::Hessian teb::Config::Optim::Solver::Nonlinear-Program::hessian

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::BaseSolverNonlinear-ProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::initHessianBF-GS(), teb::SolverSQPDense::initSolverWorkspace(), and teb::SolverSQPDense::solveImpl().

16.23.2.2 struct teb::Config::Optim::Solver::NonlinearProgram::LineSearch teb::Config::Optim::Solver::Nonlinear-Program::linesearch

Referenced by teb::SolverSQPDense::initSolverWorkspace(), and teb::SolverSQPDense::solveImpl().

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

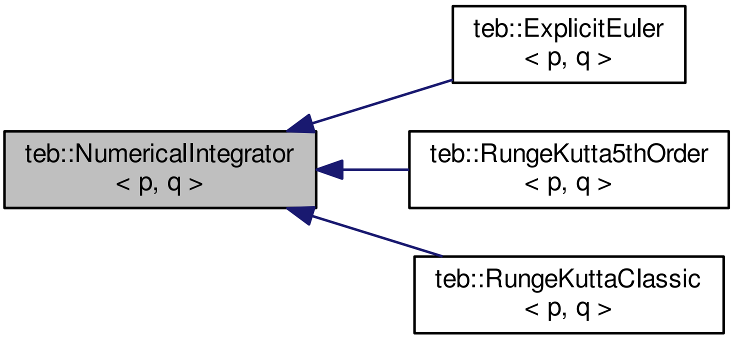
- config.h

16.24 teb::NumericalIntegrator< p, q > Class Template Reference

Interface for numerical integration methods used in simulation.

`#include <integrators.h>`

Inheritance diagram for teb::NumericalIntegrator< p, q >:



Public Types

- using StateVector = Eigen::Matrix< double, p, 1 >

    Typedef for state vector with p states [p x 1].
- using ControlVector = Eigen::Matrix< double, q, 1 >

    Typedef for control input vector with q controls [q x 1].

Public Member Functions

- NumericalIntegrator ()

    Empty Constructor.
- virtual ∼NumericalIntegrator ()

    Empty Destructor.
- virtual StateVector integrate (const Eigen::Ref< const StateVector > &x_k, const Eigen::Ref< const ControlVector > &u_k, SystemDynamics< p, q > ∗system_equation, double dt) const =0

    Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

    .

16.24.1 Detailed Description

template<int p, int q>class teb::NumericalIntegrator< p, q >

This class defines the interface for numerical integration schemes.

Todo Implement and test Runge Kutta with derivatives http://www.emis.de/journals/EJD-E/conf-proc/02/g1/goeken.pdf

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Template Parameters

p	Number of state variables
q	Number of input variables

Definition at line 36 of file integrators.h.

16.24.2 Member Typedef Documentation

16.24.2.1 template< int p, int q > using teb::NumericalIntegrator< p, q >::ControlVector = Eigen::Matrix<double,q,1>

Definition at line 43 of file integrators.h.

16.24.2.2 template< int p, int q > using teb::NumericalIntegrator< p, q >::StateVector = Eigen::Matrix<double,p,1>

Definition at line 41 of file integrators.h.

16.24.3 Constructor & Destructor Documentation

16.24.3.1 template< int p, int q > teb::NumericalIntegrator< p, q >::NumericalIntegrator ( ) [inline]

Definition at line 45 of file integrators.h.

16.24.3.2 template< int p, int q > virtual teb::NumericalIntegrator< p, q >::~NumericalIntegrator ( ) [inline], [virtual]

Definition at line 46 of file integrators.h.

16.24.4 Member Function Documentation

16.24.4.1 template< int p, int q > virtual StateVector teb::NumericalIntegrator< p, q >::integrate (const Eigen::Ref< const StateVector > & *x_k*, const Eigen::Ref< const ControlVector > & *u_k*, SystemDynamics< p, q > * *system_equation*, double *dt*) const [pure virtual]

Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

w.r.t. $\mathbf{x}$. In particular, it relies on discrete states. The method calculates $\mathbf{x}_{k+1}$ based on $\mathbf{x}_k$, $\mathbf{u}_k$ and the duration ΔT. The dedicated system dynamics equations may be specified using a SystemDynamics object.

Parameters

x_k	Current state vector $\mathbf{x}_k$ [p x 1]
u_k	Current state vector $\mathbf{u}_k$ [q x 1]
system_equation	Pointer to the SystemDynamics object that stores the system equations.
dt	Time interval for the integration

Returns

Integrated state vector $\mathbf{x}_{k+1}$ [p x 1]

Implemented in teb::RungeKutta5thOrder< p, q >, teb::RungeKuttaClassic< p, q >, and teb::ExplicitEuler< p, q >.

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

- integrators.h

16.25 teb::Config::Optim Struct Reference

Configurations related to the trajectory optimization.

```
#include <config.h>
```

Classes

- struct Solver

  Configurations related to solvers.

Public Attributes

- struct teb::Config::Optim::Solver solver

  Configurations related to solvers.
- bool force_rebuild_optim_graph = false

  Disable hot-starting from previous graph structers even if no changes to the graph are made.

16.25.1 Detailed Description

Definition at line 75 of file config.h.

16.25.2 Member Data Documentation

16.25.2.1 bool teb::Config::Optim::force_rebuild_optim_graph = false

Definition at line 133 of file config.h.

16.25.2.2 struct teb::Config::Optim::Solver teb::Config::Optim::solver

Examples:

integrator_system_classic_mpc.cpp, linear_system_ode_ctrl_comparison.cpp, mobile_robot_teb.cpp, and rocket_system.cpp.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), and teb::SolverSQPDense::solveImpl().

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

- config.h

16.26 teb::PlotOptions Struct Reference

Customize plots and figures.

```
#include <teb_plotter.h>
```

Public Attributes

- std::string title = ""

  Window title.
- bool legend = false

  Show legend if `true`*.*
- std::vector< std::string > legend_entries

  Vector of strings defining the names for the legend.
- bool skip_last_value_right_column = false
- std::vector< std::string > ylabels

  < If `true`*, the last value of the right column for each plot will be ignored (use to supress plotting the invalid control for time n)*

16.26.1 Detailed Description

This option class can be used to customize figures and plots, if they are supported by the plot function.

Todo Extend functionality

Author

Christoph Rösmann (`christoph.roesmann@tu-dortmund.de`)

Definition at line 27 of file teb_plotter.h.

16.26.2 Member Data Documentation

16.26.2.1 bool teb::PlotOptions::legend = false

Definition at line 30 of file teb_plotter.h.

Referenced by teb::TebPlotter::plotTwoCol(), teb::Simulator< p, q >::simClosedLoop(), teb::Simulator< p, q >::simOpenAndClosedLoop(), and teb::Simulator< p, q >::simOpenLoop().

16.26.2.2 std::vector<std::string> teb::PlotOptions::legend_entries

Remarks

legend should be enabled.

Definition at line 31 of file teb_plotter.h.

Referenced by teb::TebPlotter::plotTwoCol(), teb::Simulator< p, q >::simClosedLoop(), teb::Simulator< p, q >::simOpenAndClosedLoop(), and teb::Simulator< p, q >::simOpenLoop().

16.26.2.3 bool teb::PlotOptions::skip_last_value_right_column = false

Definition at line 32 of file teb_plotter.h.

Referenced by teb::TebPlotter::plotTwoCol(), teb::Simulator< p, q >::simClosedLoop(), teb::Simulator< p, q >::simOpenAndClosedLoop(), and teb::Simulator< p, q >::simOpenLoop().

16.26.2.4 std::string teb::PlotOptions::title = ""

Definition at line 29 of file teb_plotter.h.

Referenced by teb::TebPlotter::plotTwoCol(), teb::Simulator< p, q >::simClosedLoop(), teb::Simulator< p, q >::simOpenAndClosedLoop(), and teb::Simulator< p, q >::simOpenLoop().

16.26.2.5 std::vector<std::string> teb::PlotOptions::ylabels

Vector of strings to define the y labels (Column-Major Order) - leave blank in order to use default names

Definition at line 33 of file teb_plotter.h.

Referenced by teb::TebPlotter::plotTwoCol().

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

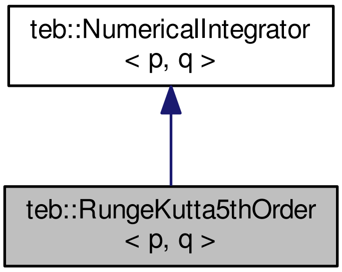
- teb_plotter.h

16.27 teb::RungeKutta5thOrder< p, q > Class Template Reference

Runge Kutta method (fifth order, slighly modified)

`#include <integrators.h>`

Inheritance diagram for teb::RungeKutta5thOrder< p, q >:



Public Types

- using StateVector = typename NumericalIntegrator< p, q >::StateVector

  Typedef for state vector with p states [p x 1].
- using ControlVector = typename NumericalIntegrator< p, q >::ControlVector

  Typedef for control input vector with q controls [q x 1].

Public Member Functions

- RungeKutta5thOrder ()

  Empty Constructor.
- virtual ∼RungeKutta5thOrder ()

Empty Destructor.

- virtual StateVector integrate (const Eigen::Ref< const StateVector > &x_k, const Eigen::Ref< const ControlVector > &u_k, SystemDynamics< p, q > *system_equation, double dt) const

  Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

  .

16.27.1 Detailed Description

template<int p, int q>class teb::RungeKutta5thOrder< p, q >

For more information please refer to http://www.jstor.org/stable/pdfplus/2027775.pdf?acceptTC=true&jpdConfirm=true (Equation 2)

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Template Parameters

p	Number of state variables
q	Number of input variables

Definition at line 157 of file integrators.h.

16.27.2 Member Typedef Documentation

16.27.2.1 template<int p, int q> using teb::RungeKutta5thOrder< p, q >::ControlVector = typename NumericalIntegrator<p,q>::ControlVector

Definition at line 163 of file integrators.h.

16.27.2.2 template<int p, int q> using teb::RungeKutta5thOrder< p, q >::StateVector = typename NumericalIntegrator<p,q>::StateVector

Definition at line 161 of file integrators.h.

16.27.3 Constructor & Destructor Documentation

16.27.3.1 template<int p, int q> teb::RungeKutta5thOrder< p, q >::RungeKutta5thOrder ( ) [inline]

Definition at line 166 of file integrators.h.

16.27.3.2 template<int p, int q> virtual teb::RungeKutta5thOrder< p, q >::~RungeKutta5thOrder ( ) [inline], [virtual]

Definition at line 167 of file integrators.h.

16.27.4 Member Function Documentation

16.27.4.1 template<int p, int q> virtual StateVector teb::RungeKutta5thOrder< p, q >::integrate (const Eigen::Ref< const StateVector > & *x_k*, const Eigen::Ref< const ControlVector > & *u_k*, SystemDynamics< p, q > * *system_equation*, double *dt*) const [inline], [virtual]

Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

w.r.t. $\mathbf{x}$. In particular, it relies on discrete states. The method calculates $\mathbf{x}_{k+1}$ based on $\mathbf{x}_k$, $\mathbf{u}_k$ and the duration ΔT. The dedicated system dynamics equations may be specified using a SystemDynamics object.

Parameters

x_k	Current state vector $\mathbf{x}_k$ [p x 1]
u_k	Current state vector $\mathbf{u}_k$ [q x 1]
system_equation	Pointer to the SystemDynamics object that stores the system equations.
dt	Time interval for the integration

Returns

Integrated state vector $\mathbf{x}_{k+1}$ [p x 1]

Implements teb::NumericalIntegrator< p, q >.

Definition at line 171 of file integrators.h.

References teb::SystemDynamics< p, q >::stateSpaceModel().

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

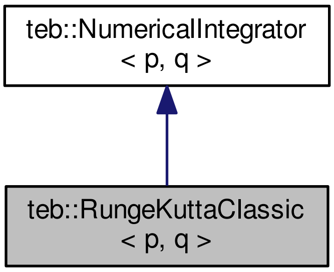
- integrators.h

16.28 teb::RungeKuttaClassic< p, q > Class Template Reference

Classic Runge Kutta method (fourth order)

`#include <integrators.h>`

Inheritance diagram for teb::RungeKuttaClassic< p, q >:



Public Types

- using StateVector = typename NumericalIntegrator< p, q >::StateVector

Typedef for state vector with p states [p x 1].

- using ControlVector = typename NumericalIntegrator< p, q >::ControlVector

  Typedef for control input vector with q controls [q x 1].

Public Member Functions

- RungeKuttaClassic ()

  Empty Constructor.

- virtual ~RungeKuttaClassic ()

  Empty Destructor.

- virtual StateVector integrate (const Eigen::Ref< const StateVector > &x_k, const Eigen::Ref< const ControlVector > &u_k, SystemDynamics< p, q > *system_equation, double dt) const

  Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

  .

16.28.1 Detailed Description

template<int p, int q>class teb::RungeKuttaClassic< p, q >

For more information please refer to http://en.wikipedia.org/wiki/Runge–Kutta_methods

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Template Parameters

p	Number of state variables
q	Number of input variables

Definition at line 115 of file integrators.h.

16.28.2 Member Typedef Documentation

16.28.2.1 template<int p, int q> using teb::RungeKuttaClassic< p, q >::ControlVector = typename NumericalIntegrator<p,q>::ControlVector

Definition at line 121 of file integrators.h.

16.28.2.2 template<int p, int q> using teb::RungeKuttaClassic< p, q >::StateVector = typename NumericalIntegrator<p,q>::StateVector

Definition at line 119 of file integrators.h.

16.28.3 Constructor & Destructor Documentation

16.28.3.1 template<int p, int q> teb::RungeKuttaClassic< p, q >::RungeKuttaClassic ( ) [inline]

Definition at line 124 of file integrators.h.

16.28.3.2 template< int p, int q > virtual teb::RungeKuttaClassic< p, q >::~RungeKuttaClassic ( ) [inline], [virtual]

Definition at line 125 of file integrators.h.

16.28.4 Member Function Documentation

16.28.4.1 template< int p, int q > virtual StateVector teb::RungeKuttaClassic< p, q >::integrate (const Eigen::Ref< const StateVector > & *x_k*, const Eigen::Ref< const ControlVector > & *u_k*, SystemDynamics< p, q > * *system_equation*, double *dt*) const [inline], [virtual]

Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

w.r.t. $\mathbf{x}$. In particular, it relies on discrete states. The method calculates $\mathbf{x}_{k+1}$ based on $\mathbf{x}_k$, $\mathbf{u}_k$ and the duration ΔT. The dedicated system dynamics equations may be specified using a SystemDynamics object.

Parameters

x_k	Current state vector $\mathbf{x}_k$ [p x 1]
u_k	Current state vector $\mathbf{u}_k$ [q x 1]
system_equation	Pointer to the SystemDynamics object that stores the system equations.
dt	Time interval for the integration

Returns

Integrated state vector $\mathbf{x}_{k+1}$ [p x 1]

Implements teb::NumericalIntegrator< p, q >.

Definition at line 129 of file integrators.h.

References teb::SystemDynamics< p, q >::stateSpaceModel().

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

- integrators.h

16.29 teb::SimResults Class Reference

Simulation results are stored and combined in this container class.

```
#include <simulator.h>
```

Classes

- struct TimeSeries

  Store measurements of dynamic systems (states and control inputs) w.r.t.

Public Attributes

- std::vector< TimeSeries > series

  Container for multiple time series (e.g. to perform store different experiments or closed- and open-loop sim results).

Protected Member Functions

- void conservativeResize (unsigned int id, int p, int q, int n)

  Resize time series, but preserve data with indices below n.
- void allocateMemory (unsigned int id, int p, int q, int n, bool set_zero=true)

  Allocate memory for all matrices and vectors of the underlying TimeSeries object.

Static Protected Member Functions

- static void conservativeResize (TimeSeries &data, int p, int q, int n)

  Resize time series, but preserve data with indices below n.
- static void allocateMemory (TimeSeries &data, int p, int q, int n, bool set_zero=true)

  Allocate memory for all matrices and vectors of the underlying TimeSeries object.

Friends

- template<int pf, int qf>
  class Simulator

16.29.1 Detailed Description

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Template Parameters

p	Number of state variables
q	Number of input variables

Definition at line 26 of file simulator.h.

16.29.2 Member Function Documentation

16.29.2.1 static void teb::SimResults::allocateMemory (TimeSeries & *data*, int *p*, int *q*, int *n*, bool *set_zero* = `true`) `[inline]`, `[static]`, `[protected]`

This method can be used to resize the underlying matrices without preserving data.

See Also

conservativeResize()

Parameters

data	TimeSeries object for which the memory should be allocated.
p	Size of the state vector.
q	Size of the control input vector.
n	Number of discrete samples.
set_zero	if `true`, all matrices and vectors are initialized to zero.

Definition at line 89 of file simulator.h.

References teb::SimResults::TimeSeries::controls, teb::SimResults::TimeSeries::states, and teb::SimResults::-TimeSeries::time.

Referenced by allocateMemory(), and teb::Simulator< p, q >::simOpenLoop().

16.29.2.2 void teb::SimResults::allocateMemory (unsigned int *id*, int *p*, int *q*, int *n*, bool *set_zero* = `true`) `[inline]`, `[protected]`

This method can be used to resize the underlying matrices without preserving data.

See Also

conservativeResize()

Parameters

id	Id/index of the time series object stored in SimResults::series
p	Size of the state vector.
q	Size of the control input vector.
n	Number of discrete samples.
set_zero	if `true`, all matrices and vectors are initialized to zero.

Definition at line 118 of file simulator.h.

References allocateMemory(), and series.

16.29.2.3 static void teb::SimResults::conservativeResize (TimeSeries & *data*, int *p*, int *q*, int *n*) `[inline]`, `[static]`, `[protected]`

Parameters

data	TimeSeries object which should be resized.
p	Size of the state vector.
q	Size of the control input vector.
n	New number of discrete samples.

Definition at line 56 of file simulator.h.

References teb::SimResults::TimeSeries::controls, teb::SimResults::TimeSeries::states, and teb::SimResults::-TimeSeries::time.

Referenced by conservativeResize(), and teb::Simulator< p, q >::simClosedLoop().

16.29.2.4 void teb::SimResults::conservativeResize (unsigned int *id*, int *p*, int *q*, int *n*) `[inline]`, `[protected]`

Parameters

id	Id/index of the time series object stored in SimResults::series
p	Size of the state vector.
q	Size of the control input vector.
n	New number of discrete samples.

Definition at line 70 of file simulator.h.

References conservativeResize(), and series.

16.29.3 Friends And Related Function Documentation

16.29.3.1 template<int pf, int qf> friend class Simulator `[friend]`

Definition at line 29 of file simulator.h.

16.29.4 Member Data Documentation

16.29.4.1 std::vector< TimeSeries > teb::SimResults::series

Definition at line 45 of file simulator.h.

Referenced by allocateMemory(), conservativeResize(), teb::Simulator< p, q >::simClosedLoop(), teb::Simulator< p, q >::simOpenAndClosedLoop(), and teb::Simulator< p, q >::simOpenLoop().

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

- simulator.h

16.30 teb::Simulator< p, q > Class Template Reference

Simulator for dynamic systems controlled by a TebController.

`#include <simulator.h>`

Public Types

- using StateVector = Eigen::Matrix< double, p, 1 >

  Typedef for state vector with p states [p x 1].
- using ControlVector = Eigen::Matrix< double, q, 1 >

  Typedef for control input vector with q controls [q x 1].
- using IntegratorPtr = std::unique_ptr< NumericalIntegrator< p, q >>

  Typedef for a smart pointer that points to the prefered numerical integration method object.

Public Member Functions

- Simulator (BaseController *controller, SystemDynamics< p, q > &system, TebPlotter *plotter=nullptr)

  Construct the simulator by accepting a TebController object, a SystemDynamics object and the TebPlotter object.
- Simulator (SystemDynamics< p, q > &system, TebPlotter *plotter=nullptr)

  Construct the simulator using a SystemDynamics object and the TebPlotter object.
- ∼Simulator ()

  Empty Destructor.
- void setSampleTime (double sample_time)

  Set the sample time for the simulation.
- double sampleTime ()

  Get the sample time of the simulator.
- void setPlotter (TebPlotter *plotter)

  Pass a TebPlotter object to the Simulator class in order to enable visualization.
- void setIntegrator (IntegratorPtr integrator)

  Set a custom numerical integration method for simulation as subclass of NumericalIntegrator.
- void setIntegrator (NumericalIntegrators int_type)

  Set a predined numerical integration method for simulation (select from teb::NumericalIntegrators)
- std::unique_ptr< SimResults > simOpenLoop (const double *const x0, const double *const xf, bool plot=true, BaseController *ctrl=nullptr)

  Perform an open-loop simulation of the system in combination with the TebController.
- std::unique_ptr< SimResults > simClosedLoop (const double *const x0, const double *const xf, double sim_time, bool manual_stepping=false, bool plot_result=true, bool plot_steps=true, BaseController *ctrl=nullptr)

  Perform a closed-loop simulation of the system in combination with the TebController.
- std::unique_ptr< SimResults > simOpenAndClosedLoop (const double *const x0, const double *const xf, double sim_time, bool manual_stepping=false, bool plot_steps=true)

Perform both an open-loop and a closed-loop simulation of the system in combination with the TebController.

- void simOpenLoop (std::vector< std::pair< BaseController ∗, std::string >> &controllers, const double ∗const x0, const double ∗const xf, bool plot=true)

  Perform an open-loop simulation of the system for different TebControllers.
- void simClosedLoop (std::vector< std::pair< BaseController ∗, std::string >> &controllers, const double ∗const x0, const double ∗const xf, double sim_time, bool plot_result=true, bool plot_steps=true)

  Perform a closed-loop simulation of the system for different TebControllers.
- StateVector systemStep (const Eigen::Ref< const StateVector > &x_k, const Eigen::Ref< const ControlVector > &u_k, double dt)
- void setControlInputSaturation (const double ∗u_min, const double ∗u_max)

  Set bounds on the control inputs for simulatin (saturation).
- void setPreStepCallback (const std::function< void(BaseController ∗, SystemDynamics< p, q > ∗, Simulator ∗)> &callback)

  Set a pointer to a function that is called at the beginning of each control step.
- void setPostStepCallback (const std::function< void(BaseController ∗, SystemDynamics< p, q > ∗, Simulator ∗)> &callback)

  Set a pointer to a function that is called after each control step.
- void setPreSimCallback (const std::function< void(BaseController ∗, SystemDynamics< p, q > ∗, Simulator ∗)> &callback)

  Set a pointer to a function that is called at the beginning of each complete simulation.

Protected Member Functions

- void plotResults (const std::vector< SimResults::TimeSeries > &time_series, teb::PlotOptions ∗options=nullptr) const

  Plot a container of SimResults::TimeSeries objects.
- void saturateControl (Eigen::Ref< ControlVector > u)

  Saturate the control inputs $\mathbf{u}$ specified by setControlInputSaturation().

Protected Attributes

- BaseController ∗ _controller = nullptr

  Pointer to the Controller used for controlling the system.
- SystemDynamics< p, q > ∗ _system = nullptr

  Pointer to the SystemDynamics object that specifies the state space equations of the dynamic system.
- IntegratorPtr _integrator = IntegratorPtr(new RungeKutta5thOrder<p,q>())

  Numerical integration method (default: RungeKutta5thOrder)
- TebPlotter ∗ _plotter = nullptr

  Pointer to a TebPlotter object used for visualization.
- double _sample_time = -1.

  Sample time for simulation (-1: inherit variable dt from teb)
- std::pair< ControlVector, ControlVector > _control_bounds = std::make_pair(ControlVector::Constant(-INF), ControlVector::Constant(INF))

  Store hard control input bounds $\mathbf{u}_{min}$ and $\mathbf{u}_{max}$ for simulation (saturation).
- std::function< void(BaseController ∗, SystemDynamics< p, q > ∗, Simulator ∗)> _callback_step_pre

  Store a pointer to a function that is called at the beginning of each control step Use this function to make user-defined changes a-priori to each step.
- std::function< void(BaseController ∗, SystemDynamics< p, q > ∗, Simulator ∗)> _callback_step_post

Store a pointer to a function that is called after each control step Use this function to make user-defined changes a-posteriori to each step.

- std::function< void(BaseController
  ∗, SystemDynamics< p, q >
  ∗, Simulator ∗)> _callback_sim_pre

  Store a pointer to a function that is called at the beginning of each simulation Use this function to make user-defined changes a-priori to the complete simulation.

Friends

- template<int pf, int qf>
  class SystemDynamics
- class SimResults

16.30.1 Detailed Description

template<int p, int q>class teb::Simulator< p, q >

This simulator supports open-loop and closed-loop simulations for systems specified using the SystemDynamics object.

Results can be visualized using the TebPlotter class.

Todo Add disturbance models.

Add reference trajectory or multiple reference steps for closed-loop control.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Template Parameters

p	Number of state variables
q	Number of input variables

Examples:

integrator_system2.cpp, integrator_system_classic_mpc.cpp, linear_system_ode_ctrl_comparison.cpp, mobile_robot_teb.cpp, and rocket_system.cpp.

Definition at line 145 of file simulator.h.

16.30.2 Member Typedef Documentation

16.30.2.1 template<int p, int q> using teb::Simulator< p, q >::ControlVector = Eigen::Matrix<double, q, 1>

Definition at line 157 of file simulator.h.

16.30.2.2 template<int p, int q> using teb::Simulator< p, q >::IntegratorPtr = std::unique_ptr<NumericalIntegrator<p,q>>

Definition at line 159 of file simulator.h.

16.30.2.3 template<int p, int q> using teb::Simulator< p, q >::StateVector = Eigen::Matrix<double,p,1>

Definition at line 155 of file simulator.h.

16.30.3 Constructor & Destructor Documentation

16.30.3.1 template<int p, int q> teb::Simulator< p, q >::Simulator (BaseController * *controller*, SystemDynamics< p, q > & *system*, TebPlotter * *plotter* = nullptr) [inline]

Parameters

controller	Reference to the Controller
system	Reference to the underlying dynamic system.
plotter	Pointer to a TebPlotter object that enables visualization.

Definition at line 169 of file simulator.h.

16.30.3.2 template<int p, int q> teb::Simulator< p, q >::Simulator (SystemDynamics< p, q > & *system*, TebPlotter * *plotter* = nullptr) [inline]

Usually you want to use the other constructor and pass a controller object. Use this constructor only, if you want to call simulations, that accept the controller object directly.

Parameters

system	Reference to the underlying dynamic system.
plotter	Pointer to a TebPlotter object that enables visualization.

Definition at line 181 of file simulator.h.

16.30.3.3 template<int p, int q> teb::Simulator< p, q >::~Simulator ( ) [inline]

Definition at line 184 of file simulator.h.

16.30.4 Member Function Documentation

16.30.4.1 template<int p, int q> void teb::Simulator< p, q >::plotResults (const std::vector< SimResults::TimeSeries > & *time_series*, teb::PlotOptions * *options* = nullptr) const [protected]

This method prepares a container of multiple time_series object in order to allow the utilization of the TebPlotter for visualization.

Customize the plot by optionally passing a PlotOptions object to this method.

Parameters

time_series	Container of time series.
options	Custom plot options.

Definition at line 390 of file simulator.hpp.

References PRINT_INFO.

16.30.4.2 template<int p, int q> double teb::Simulator< p, q >::sampleTime ( ) [inline]

See Also

setSampleTime()

Returns

Simulator sample time.

Definition at line 203 of file simulator.h.

References teb::Simulator< p, q >::_sample_time.

16.30.4.3 template<int p, int q> void **teb::Simulator**< p, q >::saturateControl (Eigen::Ref< **ControlVector** > *u*) [protected]

Parameters

u	The control input that should be bounded. The result will be stored directly to `u`.

Definition at line 373 of file simulator.hpp.

16.30.4.4 template<int p, int q> void **teb::Simulator**< p, q >::setControlInputSaturation (const double * *u_min,* const double * *u_max*) [inline]

Parameters

u_min	pointer to the minimum control bounds [q x 1]
u_max	pointer to the maximum control bounds [q x 1]

Definition at line 250 of file simulator.h.

References teb::Simulator< p, q >::_control_bounds.

16.30.4.5 template<int p, int q> void **teb::Simulator**< p, q >::setIntegrator (**IntegratorPtr** *integrator*) [inline]

Parameters

integrator	Unique pointer to a NumericalIntegrator subclass.

Definition at line 215 of file simulator.h.

References teb::Simulator< p, q >::_integrator.

16.30.4.6 template<int p, int q> void **teb::Simulator**< p, q >::setIntegrator (**NumericalIntegrators** *int_type*)

Parameters

int_type	Integrator type selected from teb::NumericalIntegrators.

Definition at line 352 of file simulator.hpp.

References teb::EXPLICIT_EULER, teb::RUNGE_KUTTA_5TH, and teb::RUNGE_KUTTA_CLASSIC.

16.30.4.7 template<int p, int q> void **teb::Simulator**< p, q >::setPlotter (**TebPlotter** * *plotter*) [inline]

Parameters

plotter	Pointer to the plotter object (no memory will be freed).

Definition at line 209 of file simulator.h.

References teb::Simulator< p, q >::_plotter.

16.30.4.8 template<int p, int q> void teb::Simulator< p, q >::setPostStepCallback (const std::function< void(BaseController *, SystemDynamics< p, q > *, Simulator< p, q > *)> & *callback*) [inline]

The callback function should have the following arguments:

- BaseController* ctrl - Pointer to the current controller object
- SystemDynamics* system - Pointer to the current system object
- Simulator* sim - Pointer to the caller (TebSimulator class) itself

  Remarks

  Use std::bind() in order to pass class methods

  Parameters

callback	function (object) that should be called

Definition at line 279 of file simulator.h.

References teb::Simulator< p, q >::_callback_step_post.

16.30.4.9 template<int p, int q> void teb::Simulator< p, q >::setPreSimCallback (const std::function< void(BaseController *, SystemDynamics< p, q > *, Simulator< p, q > *)> & *callback*) [inline]

The callback function should have the following arguments:

- BaseController* ctrl - Pointer to the current controller object
- SystemDynamics* system - Pointer to the current system object
- Simulator* sim - Pointer to the caller (TebSimulator class) itself

  Remarks

  Use std::bind() in order to pass class methods

  Parameters

callback	function (object) that should be called

Definition at line 294 of file simulator.h.

References teb::Simulator< p, q >::_callback_sim_pre.

16.30.4.10 template<int p, int q> void teb::Simulator< p, q >::setPreStepCallback (const std::function< void(BaseController *, SystemDynamics< p, q > *, Simulator< p, q > *)> & *callback*) [inline]

The callback function should have the following arguments:

- BaseController* ctrl - Pointer to the current controller object
- SystemDynamics* system - Pointer to the current system object
- Simulator* sim - Pointer to the caller (TebSimulator class) itself

Remarks

Use std::bind() in order to pass class methods

Parameters

callback	function (object) that should be called

Definition at line 265 of file simulator.h.

References teb::Simulator< p, q >::_callback_step_pre.

16.30.4.11 template<int p, int q> void **teb::Simulator**< p, q >::setSampleTime (double *sample_time*) [inline]

The system is simulated for the duration `sample_time` [sec] until a new control is applied. If `sample_time` == -1, the sample time is inherited from the TebController.

Todo Allow different sample_times for controller and plant simulation (the TEB sample time ΔT might be higher ...)

Parameters

sample_time	Specified sample time.

Definition at line 196 of file simulator.h.

References teb::Simulator< p, q >::_sample_time.

16.30.4.12 template<int p, int q> std::unique_ptr< **SimResults** > **teb::Simulator**< p, q >::simClosedLoop (const double *const *x0*, const double *const *xf*, double *sim_time*, bool *manual_stepping* = `false`, bool *plot_result* = `true`, bool *plot_steps* = `true`, **BaseController** * *ctrl* = `nullptr`)

This method performs a closed-loop simulation of the system controlled by the TebSimulator, both specified in the constructor of this class.

The TEB optimization is solved in each sampling-interval specified by setSampleTime() to determine the next optimal control for the point to point transition from the most recent measurement of the state vector $\mathbf{x}_k$ to the final state vector $\mathbf{x}_f$. As a starting point, the first state vector is given by $\mathbf{x}_0$.

Bug If we set a title or legend in the plot options for stepping (!), than the plot won't be rendered correctly.

Parameters

x0	Double array with *p* values representing the start state vector
xf	Double array with *p* values representing the final state vector
sim_time	Total simulation time (duration of the simulation)
manual_-stepping	if `true`, the simulation is paused after each sampling interval. During pause, the user can type in the desired number of steps until the next pause will be instantiated. If the user chooses "-1" steps, the simulation will be proceeded until sim_time is exceeded.
plot_result	if `true`, the results at the end of the simulation are plotted into a figure using TebPlotter.
plot_steps	if `true`, the results after each sample_inteveral are plotted into a figure using TebPlotter (shows the progress of the control).
ctrl	Pointer to an external controller: set to nullptr in order to use the class member _controller.

Returns

SimResults containing the time series of the closed-loop control.

Definition at line 119 of file simulator.hpp.

References teb::SimResults::conservativeResize(), teb::BaseController::firstControl(), teb::BaseController::getDt(), INPUT_STREAM, teb::PlotOptions::legend, teb::PlotOptions::legend_entries, PRINT_INFO, PRINT_INFO_ONCE, teb::BaseController::resetController(), teb::SimResults::series, teb::PlotOptions::skip_last_value_right_column, teb::BaseController::step(), and teb::PlotOptions::title.

16.30.4.13 template<int p, int q> void **teb::Simulator**< p, q >::simClosedLoop (std::vector< std::pair< **BaseController** ∗, std::string >> & *controllers*, const double ∗const *x0*, const double ∗const *xf*, double *sim_time*, bool *plot_result* = `true`, bool *plot_steps* = `true`)

This method performs an closed-loop simulation of the system controlled by different Controllers passed to this method.

The system is specified using the constructor of this class.

The TEB optimization is solved in each sampling-interval specified by setSampleTime() to determine the next optimal control for the point to point transition from the most recent measurement of the state vector $\mathbf{x}_k$ to the final state vector $\mathbf{x}_f$. As a starting point, the first state vector is given by $\mathbf{x}_0$.

Parameters

controllers	array of different TebControllers with augmented name as string
x0	Double array with *p* values representing the start state vector
xf	Double array with *p* values representing the final state vector
sim_time	Total simulation time (duration of the simulation)
plot_result	if `true`, the results at the end of the simulation are plotted into a figure using TebPlotter.
plot_steps	if `true`, the results after each sample_inteveral are plotted into a figure using TebPlotter (shows the progress of the control).

Definition at line 328 of file simulator.hpp.

References teb::PlotOptions::legend, teb::PlotOptions::legend_entries, teb::SimResults::series, teb::PlotOptions::skip_last_value_right_column, and teb::PlotOptions::title.

16.30.4.14 template<int p, int q> std::unique_ptr< **SimResults** > **teb::Simulator**< p, q >::simOpenAndClosedLoop (const double ∗const *x0*, const double ∗const *xf*, double *sim_time*, bool *manual_stepping* = `false`, bool *plot_steps* = `true`)

This method performs both, an open-loop simulation of the system controlled by the TebSimulator and the closed-loop simulation of the system.

Refer to simOpenLoop() and simClosedLoop() for details.

The results are combined and shown in a single figure using TebPlotter.

Parameters

x0	Double array with *p* values representing the start state vector
xf	Double array with *p* values representing the final state vector
sim_time	Total simulation time for the closed-loop simulation (duration of the simulation)
manual_-stepping	if `true`, the closed-loop simulation is paused after each sampling interval. During pause, the user can type in the desired number of steps until the next pause will be instantiated. If the user chooses "-1" steps, the simulation will be proceeded until sim_time is exceeded.
plot_steps	if `true`, the results after each sample_inteveral of the closed-loop control are plotted into a figure using TebPlotter (shows the progress of the control).

Returns

SimResults containing the time series of the open-loop control, the closed-loop control and the states and controls planned by the controller.

Definition at line 249 of file simulator.hpp.

References teb::PlotOptions::legend, teb::PlotOptions::legend_entries, teb::SimResults::series, teb::PlotOptions::skip_last_value_right_column, and teb::PlotOptions::title.

16.30.4.15 template<int p, int q> std::unique_ptr< **SimResults** > **teb::Simulator**< p, q >::simOpenLoop (const double *const *x0*, const double *const *xf*, bool *plot* = `true`, **BaseController** * *ctrl* = `nullptr`)

This method performs an open-loop simulation of the system controlled by the TebSimulator, both specified in the constructor of this class.

The TEB optimization is solved once to determine the optimal control for the point to point transition from the start state vector $\mathbf{x}_0$ to the final state vector $\mathbf{x}_f$.

The determined control is applied to the simulated plant. In the abscence of model mismatch and disturbances the output should be similar to the planned one by the TEB, if the system dynamics equations are satisfied.

Parameters

x0	Double array with *p* values representing the start state vector
xf	Double array with *p* values representing the final state vector
plot	if `true`, the results are plotted into a figure using TebPlotter.
ctrl	Pointer to an external controller: set to nullptr in order to use the class member _controller.

Returns

SimResults containing the time series of the open-loop control and the states and controls planned by the controller.

Definition at line 28 of file simulator.hpp.

References teb::SimResults::allocateMemory(), teb::BaseController::firstState(), teb::BaseController::getAbsoluteTimeVec(), teb::BaseController::getDt(), teb::BaseController::getN(), teb::BaseController::getStateCtrlInfoMat(), teb::PlotOptions::legend, teb::PlotOptions::legend_entries, teb::BaseController::resetController(), teb::BaseController::returnControlInputSequence(), teb::SimResults::series, teb::BaseController::step(), and teb::PlotOptions::title.

16.30.4.16 template<int p, int q> void **teb::Simulator**< p, q >::simOpenLoop (std::vector< std::pair< **BaseController** *, std::string >> & *controllers*, const double *const *x0*, const double *const *xf*, bool *plot* = `true`)

This method performs an open-loop simulation of the system controlled by different Controllers passed to this method.

The system is specified using the constructor of this class.

The TEB optimization is solved once to determine the optimal control for the point to point transition from the start state vector $\mathbf{x}_0$ to the final state vector $\mathbf{x}_f$.

Parameters

controllers	array of different TebControllers with augmented name as string
x0	Double array with *p* values representing the start state vector
xf	Double array with *p* values representing the final state vector
plot	if `true`, the results are plotted into a figure using TebPlotter.

Definition at line 289 of file simulator.hpp.

References teb::PlotOptions::legend, teb::PlotOptions::legend_entries, teb::SimResults::series, teb::PlotOptions::skip_last_value_right_column, and teb::PlotOptions::title.

16.30.4.17 template<int p, int q> StateVector teb::Simulator< p, q >::systemStep (const Eigen::Ref< const StateVector > & *x_k*, const Eigen::Ref< const ControlVector > & *u_k*, double *dt*) [inline]

Definition at line 238 of file simulator.h.

References teb::Simulator< p, q >::_integrator, and teb::Simulator< p, q >::_system.

16.30.5 Friends And Related Function Documentation

16.30.5.1 template<int p, int q> friend class SimResults [friend]

Definition at line 150 of file simulator.h.

16.30.5.2 template<int p, int q> template<int pf, int qf> friend class SystemDynamics [friend]

Definition at line 148 of file simulator.h.

16.30.6 Member Data Documentation

16.30.6.1 template<int p, int q> std::function<void(BaseController*,SystemDynamics<p,q>*, Simulator*)> teb::Simulator< p, q >::_callback_sim_pre [protected]

Definition at line 334 of file simulator.h.

Referenced by teb::Simulator< p, q >::setPreSimCallback().

16.30.6.2 template<int p, int q> std::function<void(BaseController*,SystemDynamics<p,q>*, Simulator*)> teb::Simulator< p, q >::_callback_step_post [protected]

Definition at line 329 of file simulator.h.

Referenced by teb::Simulator< p, q >::setPostStepCallback().

16.30.6.3 template<int p, int q> std::function<void(BaseController*,SystemDynamics<p,q>*, Simulator*)> teb::Simulator< p, q >::_callback_step_pre [protected]

Definition at line 324 of file simulator.h.

Referenced by teb::Simulator< p, q >::setPreStepCallback().

16.30.6.4 template<int p, int q> std::pair<ControlVector,ControlVector> teb::Simulator< p, q >::_control_bounds = std::make_pair(ControlVector::Constant(-INF), ControlVector::Constant(INF)) [protected]

Definition at line 318 of file simulator.h.

Referenced by teb::Simulator< p, q >::setControlInputSaturation().

16.30.6.5 template<int p, int q> BaseController* teb::Simulator< p, q >::_controller = nullptr [protected]

Definition at line 310 of file simulator.h.

16.30.6.6 template< int p, int q> IntegratorPtr teb::Simulator< p, q >::_integrator = IntegratorPtr(new RungeKutta5thOrder<p,q>()) [protected]

Definition at line 312 of file simulator.h.

Referenced by teb::Simulator< p, q >::setIntegrator(), and teb::Simulator< p, q >::systemStep().

16.30.6.7 template< int p, int q> TebPlotter* teb::Simulator< p, q >::_plotter = nullptr [protected]

Definition at line 314 of file simulator.h.

Referenced by teb::Simulator< p, q >::setPlotter().

16.30.6.8 template< int p, int q> double teb::Simulator< p, q >::_sample_time = -1. [protected]

Definition at line 315 of file simulator.h.

Referenced by teb::Simulator< p, q >::sampleTime(), and teb::Simulator< p, q >::setSampleTime().

16.30.6.9 template< int p, int q> SystemDynamics<p,q>* teb::Simulator< p, q >::_system = nullptr [protected]

Definition at line 311 of file simulator.h.

Referenced by teb::Simulator< p, q >::systemStep().

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

- simulator.h
- simulator.hpp

16.31 teb::Config::Optim::Solver Struct Reference

Configurations related to solvers.

`#include <config.h>`

Classes

- struct Lsq

  Configurations especially for least-squares-solvers.
- struct Nlopt

  Configurations especially for the nlopt solver wrapper.
- struct NonlinearProgram

  Settings for constrained optimiziation solver (nonlinear program solver)

Public Attributes

- unsigned int solver_iter = 5
- struct
  teb::Config::Optim::Solver::Lsq lsq

  Configurations especially for least-squares-solvers.
- struct
  teb::Config::Optim::Solver::NonlinearProgram nonlin_prog

Settings for constrained optimiziation solver (nonlinear program solver)

- struct
  teb::Config::Optim::Solver::Nlopt nlopt

16.31.1 Detailed Description

Definition at line 78 of file config.h.

16.31.2 Member Data Documentation

16.31.2.1 struct **teb::Config::Optim::Solver::Lsq** teb::Config::Optim::Solver::lsq

Examples:

integrator_system_classic_mpc.cpp, linear_system_ode_ctrl_comparison.cpp, mobile_robot_teb.cpp, and rocket_system.cpp.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), and teb::BaseSolverLeastSquares::buildValueVector().

16.31.2.2 struct **teb::Config::Optim::Solver::Nlopt** teb::Config::Optim::Solver::nlopt

16.31.2.3 struct **teb::Config::Optim::Solver::NonlinearProgram** teb::Config::Optim::Solver::nonlin_prog

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), and teb::SolverSQPDense::solveImpl().

16.31.2.4 unsigned int teb::Config::Optim::Solver::solver_iter = 5

Definition at line 80 of file config.h.

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

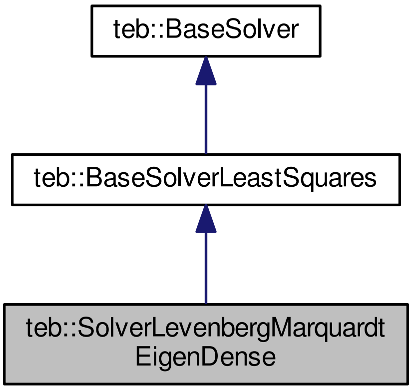
- config.h

16.32 teb::SolverLevenbergMarquardtEigenDense Class Reference

Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Dense matrices version).

`#include <solver_levenbergmarquardt_eigen_dense.h>`

Inheritance diagram for teb::SolverLevenbergMarquardtEigenDense:



Public Types

- using VertexContainer = HyperGraph::VertexContainer

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).
- using EdgeContainer = HyperGraph::EdgeContainer

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).

Public Member Functions

- SolverLevenbergMarquardtEigenDense ()

  Empty Constructor.
- void setConfig (const Config ∗config)

  Set config including solver settings.
- virtual bool solve (HyperGraph ∗optimizable_graph)

  Solve an optimization problem defined by an hyper-graph.

Public Attributes

- const Config ∗ cfg = nullptr

  Store pointer to Config object.
- EIGEN_MAKE_ALIGNED_OPERATOR_NEW

Protected Member Functions

- virtual bool solveImpl ()

  Solve the actual optimization problem.
- void buildJacobian ()

  Construct the jacobian of the complete least-squares optimziation problem.
- virtual void initWorkspaces ()

  Initialize workspaces for the block jacobians and hessians.

- void buildValueVector ()

  Build value / cost vector including all objectives and constraints.
- int getValueDimension () const

  Get dimension of the composed value/cost vector (including objectives and constraints).
- double getChi2 () const

  Calculate the χ^2 error of the optimization problem.
- void adaptWeights ()

  Automatic weight adaptation that increases soft constraint weights after each outer teb iteration.
- void applyIncrement (const Eigen::Ref< const Eigen::VectorXd > &delta)

  Apply a new increment optained by a local optimization to the active vertices.
- void applyOptVec (const Eigen::Ref< const Eigen::VectorXd > &opt_vec)

  Overwrite TEB states and control inputs with new values.
- Eigen::VectorXd getOptVecCopy () const

  Get a copy of the optimization vector (TEB states, control inputs and dt)
- int getOptVecDimension () const

  Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).
- int getDimObjectives () const

  Get dimension of the objective function.
- int getDimEqualities () const

  Get dimension of the equality constraints.
- int getDimInequalities () const

  Get dimension of the inequality constraints.
- void backupVertices ()

  Backup all active vertices.
- void restoreVertices ()

  Restore all values of active vertices from the backup stacks.
- void restoreVerticesButKeepBackup ()

  Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.
- void discardBackupVertices ()

  Discard all values of active vertices from the backup stacks.

Protected Attributes

- Eigen::MatrixXd _jacobian

  Store the jacobian matrix of the complete least-squares optimziation problem.
- Eigen::VectorXd _values

  Store the value vector computed in buildValueVector().
- int _weight_adapt_count = 0

  Store current state of the weight adaptation method adaptWeights().
- double _weight_equalities = 1

  Store current weight for equality soft-constraints.
- double _weight_inequalities = 1

  Store current weight for inequality soft-constraints.
- int _val_dim = -1

  Store dimension of the value vector here (see getValueDimension()).
- VertexContainer ∗ _active_vertices = nullptr

  Pointer to active vertex container (active = non-fixed)
- EdgeContainer ∗ _objectives = nullptr

  Pointer to edges representing objective functions.
- EdgeContainer ∗ _equalities = nullptr

Pointer to edges representing equality constraints.

- EdgeContainer ∗ _inequalities = nullptr

  Pointer to edges representing inequality constraints.
- int _opt_vec_dim = -1

  Store dimension of the optimziation vector here (see getOptVecDimension()).
- int _objective_dim = -1

  Store dimension of the objective function (see getDimObjectives()).
- int _equalities_dim = -1

  Store dimension of the equality constraints (see getDimEqualities()).
- int _inequalities_dim = -1

  Store dimension of the inequality constraints (see getDimInequalities()).
- unsigned int _no_vert_backups = 0

  Track number of vertices backups made.
- bool _graph_structure_modified = true

  Mark if a new graph structure is available, thus we need to reinitialze all workspaces.

16.32.1 Detailed Description

This solver implements the Levenberg-Marquardt algorithm also implemented in [2]. The underlying linear system is solved using Eigens Dense Matrix algebra (`http://eigen.tuxfamily.org/dox/group__Tutorial-LinearAlgebra.html`).

The jacobian required for the optimization routine is stored in a dense matrix. Therefore no exploitation of the sparse structure for solving the linear system is taken into account.

Author

Christoph Rösmann (`christoph.roesmann@tu-dortmund.de`)

Definition at line 25 of file solver_levenbergmarquardt_eigen_dense.h.

16.32.2 Member Typedef Documentation

16.32.2.1 using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer [inherited]

Definition at line 36 of file base_solver.h.

16.32.2.2 using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer [inherited]

Definition at line 34 of file base_solver.h.

16.32.3 Constructor & Destructor Documentation

16.32.3.1 teb::SolverLevenbergMarquardtEigenDense::SolverLevenbergMarquardtEigenDense ( ) [inline]

Definition at line 30 of file solver_levenbergmarquardt_eigen_dense.h.

16.32.4 Member Function Documentation

16.32.4.1 void teb::BaseSolverLeastSquares::adaptWeights ( ) [protected], [inherited]

This method increases the soft constraint weights for inequalities and equalities after each outer TEB optimization loop (see TebController::optimizeTEB()).

After the outer TEB loop is completed (Config::Teb::teb_iter), the weights are set back to Config::Optim::Solver::-Lsq::weight_equalities and Config::Optim::Solver::Lsq::weight_inequalities.

The weights are increased by $\sigma = \sigma_{init} \cdot \gamma^i$. γ denotes the increasement factor Config::Optim::Solver::Lsq::weight_-adaptation_factor. i denotes the current iteration number. σ_{init} is obtained from the config (see above).

Call this method at the beginning of solveImpl()!

Remarks

This procedere forces the optimization result to first contract the trajectory in sense of the objective, and afterwards trying to satisfy the constraints. Starting with very high weights in advance could lead to abrupt gradients for the solver. In addition the effect for the objectives (e.g. time optimallity) could be slow, if the corresponding gradients are extremly small in comparision to constraint gradients.

See Also

TebController::optimizeTEB()

Definition at line 100 of file base_solver_least_squares.cpp.

References teb::BaseSolverLeastSquares::_weight_adapt_count, teb::BaseSolverLeastSquares::_weight_-equalities, teb::BaseSolverLeastSquares::_weight_inequalities, teb::BaseSolver::cfg, teb::Config::Optim::Solver-::lsq, teb::Config::optim, teb::Config::Optim::solver, teb::Config::teb, teb::Config::Teb::teb_iter, teb::Config::Optim::-Solver::Lsq::weight_adaptation_factor, teb::Config::Optim::Solver::Lsq::weight_equalities, and teb::Config::Optim::-Solver::Lsq::weight_inequalities.

Referenced by solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.32.4.2 void teb::BaseSolver::applyIncrement (const Eigen::Ref< const Eigen::VectorXd > & *delta*) `[inline]`, `[protected]`, `[inherited]`

This method iterates the active vertices container and calls VertexType::plusFree() for each vertex.

Parameters

delta	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.

Definition at line 127 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense-::solveImpl().

16.32.4.3 void teb::BaseSolver::applyOptVec (const Eigen::Ref< const Eigen::VectorXd > & *opt_vec*) `[inline]`, `[protected]`, `[inherited]`

This method iterates the active vertices container and calls VertexType::setFree() for each vertex.

Parameters

opt_vec	Eigen::Vector containing all values. The length equals the dimension of all free variables.

Definition at line 144 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

16.32.4.4 void teb::BaseSolver::backupVertices ( ) `[inline]`, `[protected]`, `[inherited]`

See Also

VertexType::push()

Definition at line 223 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.32.4.5 void teb::SolverLevenbergMarquardtEigenDense::buildJacobian ( ) [protected]

This method constructs the jacobian of the complete optimziation problem.

The size of the jacobian is [getValueDimension() x getOptVecDimension()]. This method queries the EdgeType::computeJacobian() of each edge. The column index for each entry is determined using TebVertex::getOptVecIdx and the number of free dimensions for each vertex.

Since this solver relies on soft-constraints for equality and inequality constraints, the chain-rule for calculating derivatives is utilized. Equality constraints are treaten as usual objective, therefore the jacobian does not need to be changed.

For inequality constraints, the derivative of $\max\left(c(x),0\right)$ is

$$\begin{cases} \frac{\partial}{\partial x} c_i(x) & \text{if } c_i(x) > 0 \\ 0 & \text{otherwise} \end{cases} = \mathbf{J}_{ineq}(:, j=1:n) .* \frac{1}{\mathbf{c}(x)} .* \max\left(\mathbf{c}(x), \mathbf{0}\right)$$

The max operator is applicated component-wise. ' $.*$' corresponds to Matlab's component-wise multiplication.

The resulting jacobian matrix is stored to SolverLevenbergMarquardtEigenDense::_jacobian;

Definition at line 157 of file solver_levenbergmarquardt_eigen_dense.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_inequalities, _jacobian, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::BaseSolverLeastSquares::_val_dim, teb::BaseSolverLeastSquares::_weight_equalities, teb::BaseSolverLeastSquares::_weight_inequalities, teb::BaseSolver::cfg, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), teb::VertexType::isFixedComp(), teb::Config::Optim::Solver::lsq, teb::Config::optim, teb::Config::Optim::Solver::Lsq::soft_constr_epsilon, and teb::Config::Optim::solver.

Referenced by solveImpl().

16.32.4.6 void teb::BaseSolverLeastSquares::buildValueVector ( ) [protected], [inherited]

This method constructs the full value/cost vector $\mathbf{f}$ (not $\mathbf{f}^2$) for the underlying least-squares optimization problem.

The lenght of the vector can be obtained using getValueDimension().

Constraints are transformed into costs/objectives using soft constraints:

- Equality constraints $c(x) = 0$.

  These constraints are directly taken as objectives since $c^2(x)$ has a local minima at $x = 0$.

- Inequality constraints $c(x) \leq 0$

  These constraints are transformed using $f = \max\left(c(x),0\right)$ (component-wise).

  This notation is similar to: c(x) <= epsilon ? 0 : c(x).

  Afterwards (in the actual solveImpl() method, the cost function will be squared, that leads to twice differnetiable costs for the constraints, if f is piecewise differentiable once and if it intersectects with the abscissa.

In addition the weights BaseSolverLeastSquares::_weight_equalities and BaseSolverLeastSquares::_weight_inequalities are taken into account in order to weight the "soft constraint objectives". To make the weights comparable to [2] and our Matlab TEB version, we take the square root of both weights, since the least-square problem here is formulated as $f^2(x,\sigma)$ instead of $\sigma f^2(x)$. σ denotes the weight.

The results are stored internally to BaseSolverLeastSquares::_values.

See Also

adaptWeights(), getValueDimension()

Definition at line 31 of file base_solver_least_squares.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_inequalities, teb::BaseSolver::_objectives, teb::BaseSolverLeastSquares::_val_dim, teb::BaseSolverLeastSquares::_values, teb::BaseSolverLeastSquares::_weight_equalities, teb::BaseSolverLeastSquares::_weight_inequalities, teb::BaseSolver::cfg, teb::Config::Optim::Solver::lsq, teb::Config::optim, teb::Config::Optim::Solver::Lsq::soft_constr_epsilon, and teb::Config::Optim::solver.

Referenced by solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.32.4.7 void teb::BaseSolver::discardBackupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::discardTop()

Definition at line 229 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::BaseSolver::solve(), solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.32.4.8 double teb::BaseSolverLeastSquares::getChi2 ( ) const [inline], [protected], [inherited]

The χ^2 error is the scalar cost value of the least-squares optimization problem since the total cost function is composed of weighted terms:

$$f(\mathscr{B}) = \mathbf{f}(\mathscr{B})^T \mathbf{f}(\mathscr{B})$$

Weights are included in $\mathbf{f}(\mathscr{B})$. In this case it is $\chi^2 = f(\mathscr{B})$.

Todo Maybe switch to the formulation $f(\mathscr{B}) = \mathbf{f}(\mathscr{B})^T \Omega \mathbf{f}(\mathscr{B})$ similar to g2o and Teb-Matlab.

Definition at line 82 of file base_solver_least_squares.h.

References teb::BaseSolverLeastSquares::_values.

Referenced by solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.32.4.9 int teb::BaseSolver::getDimEqualities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file base_solver.h.

References teb::BaseSolver::_equalities.

Referenced by teb::BaseSolver::solve().

16.32.4.10 int teb::BaseSolver::getDimInequalities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file base_solver.h.

References teb::BaseSolver::_inequalities.

Referenced by teb::BaseSolver::solve().

16.32.4.11 int teb::BaseSolver::getDimObjectives ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file base_solver.h.

References teb::BaseSolver::_objectives.

Referenced by teb::BaseSolver::solve().

16.32.4.12 Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::getDataFree() for each vertex.

Remarks

Make sure BaseSolver::_opt_vec_dim is valid (see solve()).

Returns

Eigen::Vector containing all values. The length equals the dimension of all free variables (getOptVecDimension()).

Definition at line 162 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_opt_vec_dim.

16.32.4.13 int teb::BaseSolver::getOptVecDimension ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

See Also

TebController::getActiveVertices(), getValueDimension()

Definition at line 180 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::BaseSolver::solve().

16.32.4.14 int teb::BaseSolverLeastSquares::getValueDimension ( ) const [protected], [inherited]

This method returns the dimension of the full cost/value vector including all objective and constraints.

In particular it sums up all dimensions of single edges stored in the given hyper-graph.

Remarks

Make sure BaseSolver::_objective_dim, BaseSolver::_equalities_dim and BaseSolver::_inequalities_dim are valid (see solve()).

Returns

dimension of the complete value/cost vector.

See Also

buildValueVector(), getOptVecDimension(), getDimObjectives(), getDimEqualities(), getDimInequalities()

Definition at line 73 of file base_solver_least_squares.cpp.

References teb::BaseSolver::_equalities_dim, teb::BaseSolver::_inequalities_dim, and teb::BaseSolver::_objective_dim.

Referenced by solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.32.4.15 virtual void teb::BaseSolverLeastSquares::initWorkspaces ( ) [inline], [protected], [virtual], [inherited]

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each EdgeType) or map to existing memory.

Reimplemented from teb::BaseSolver.

Definition at line 43 of file base_solver_least_squares.h.

References teb::BaseSolver::_equalities, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_objectives, PRINT_DEBUG_COND_ONCE, and teb::QUADRATIC.

16.32.4.16 void teb::BaseSolver::restoreVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::pop()

Definition at line 225 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.32.4.17 void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline], [protected], [inherited]

See Also

VertexType::top()

Definition at line 227 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverSQPDense::solveImpl().

16.32.4.18 void teb::BaseSolver::setConfig (const Config * *config*) [inline], [inherited]

Remarks

This function is called from the TEB class within setSolver method.

Parameters

config	Pointer to Config object.

Definition at line 49 of file base_solver.h.

References teb::BaseSolver::cfg.

Referenced by teb::TebController< p, q >::setSolver().

16.32.4.19 virtual bool teb::BaseSolver::solve (HyperGraph * *optimizable_graph*) [inline], [virtual], [inherited]

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with solveImpl().

Todo Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

Parameters

optimizable_-graph	pointer to the HyperGraph that should be solved

Definition at line 61 of file base_solver.h.

References teb::BaseSolver::_active_vertices, teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, teb::BaseSolver::_no_vert_backups, teb::BaseSolver::_objective_dim, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::HyperGraph::activeVertices(), teb::BaseSolver::discardBackupVertices(), teb::HyperGraph::equalities(), teb::BaseSolver::getDimEqualities(), teb::BaseSolver::getDimInequalities(), teb::BaseSolver::getDimObjectives(), teb::BaseSolver::getOptVecDimension(), teb::HyperGraph::inequalities(), teb::BaseSolver::initWorkspaces(), teb::HyperGraph::isGraphModified(), teb::HyperGraph::objectives(), and teb::BaseSolver::solveImpl().

16.32.4.20 bool teb::SolverLevenbergMarquardtEigenDense::solveImpl ( ) [protected], [virtual]

Store results to the vertices of the active vertices container.

Return values

true	optimization was successfull.
false	optimization was not successfull.

Implements teb::BaseSolverLeastSquares.

Definition at line 8 of file solver_levenbergmarquardt_eigen_dense.cpp.

References _jacobian, teb::BaseSolver::_opt_vec_dim, teb::BaseSolverLeastSquares::_val_dim, teb::BaseSolverLeastSquares::_values, teb::BaseSolverLeastSquares::adaptWeights(), teb::BaseSolver::applyIncrement(), teb::BaseSolver::backupVertices(), buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolverLeastSquares::getChi2(), teb::BaseSolverLeastSquares::getValueDimension(), and teb::BaseSolver::restoreVertices().

16.32.5 Member Data Documentation

16.32.5.1 VertexContainer* teb::BaseSolver::_active_vertices = nullptr [protected], [inherited]

Definition at line 233 of file base_solver.h.

Referenced by teb::BaseSolver::applyIncrement(), teb::BaseSolver::applyOptVec(), teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::getOptVecCopy(), teb::BaseSolver::getOptVecDimension(), teb::BaseSolver::restoreVertices(), teb::BaseSolver::restoreVerticesButKeepBackup(), and teb::BaseSolver::solve().

16.32.5.2 EdgeContainer* teb::BaseSolver::_equalities = nullptr [protected], [inherited]

Definition at line 235 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), buildJacobian(), teb::SolverLevenbergMarquardt-

EigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimEqualities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and teb::BaseSolver::solve().

16.32.5.3 int teb::BaseSolver::_equalities_dim = -1 [protected], [inherited]

Definition at line 240 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::SolverNloptPackage::getEqConstrDimFromStorage(), teb::BaseSolverLeastSquares::getValueDimension(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.32.5.4 bool teb::BaseSolver::_graph_structure_modified = true [protected], [inherited]

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the solve() method.

Definition at line 250 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.32.5.5 EdgeContainer* teb::BaseSolver::_inequalities = nullptr [protected], [inherited]

Definition at line 236 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsInEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimInequalities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and teb::BaseSolver::solve().

16.32.5.6 int teb::BaseSolver::_inequalities_dim = -1 [protected], [inherited]

Definition at line 241 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::BaseSolverLeastSquares::getValueDimension(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.32.5.7 Eigen::MatrixXd teb::SolverLevenbergMarquardtEigenDense::_jacobian [protected]

Definition at line 40 of file solver_levenbergmarquardt_eigen_dense.h.

Referenced by buildJacobian(), and solveImpl().

16.32.5.8 unsigned int teb::BaseSolver::_no_vert_backups = 0 [protected], [inherited]

Definition at line 243 of file base_solver.h.

Referenced by teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::restoreVertices(), and teb::BaseSolver::solve().

16.32.5.9 int teb::BaseSolver::_objective_dim = -1 [protected], [inherited]

Definition at line 239 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::getValueDimension(), and teb::BaseSolver::solve().

16.32.5.10 EdgeContainer* teb::BaseSolver::_objectives = nullptr [protected], [inherited]

Definition at line 234 of file base_solver.h.

Referenced by buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::buildObjectiveValue(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimObjectives(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::SolverNloptPackage::objectives(), and teb::BaseSolver::solve().

16.32.5.11 int teb::BaseSolver::_opt_vec_dim = -1 [protected], [inherited]

Definition at line 238 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getOptVecCopy(), teb::SolverNloptPackage::getOptVecDimFromStorage(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.32.5.12 int teb::BaseSolverLeastSquares::_val_dim = -1 [protected], [inherited]

Definition at line 96 of file base_solver_least_squares.h.

Referenced by teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian(), buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.32.5.13 Eigen::VectorXd teb::BaseSolverLeastSquares::_values [protected], [inherited]

Definition at line 90 of file base_solver_least_squares.h.

Referenced by teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverLeastSquares::getChi2(), solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.32.5.14 int teb::BaseSolverLeastSquares::_weight_adapt_count = 0 [protected], [inherited]

Definition at line 92 of file base_solver_least_squares.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights().

16.32.5.15 double teb::BaseSolverLeastSquares::_weight_equalities = 1 [protected], [inherited]

Definition at line 93 of file base_solver_least_squares.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), and teb::BaseSolverLeastSquares::buildValueVector().

16.32.5.16 double teb::BaseSolverLeastSquares::_weight_inequalities = 1 [protected], [inherited]

Definition at line 94 of file base_solver_least_squares.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), and teb::BaseSolverLeastSquares::buildValueVector().

16.32.5.17 const Config* teb::BaseSolver::cfg = nullptr [inherited]

See Also

setConfig()

Definition at line 95 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::setConfig(), and teb::SolverSQPDense::solveImpl().

16.32.5.18 teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW [inherited]

Definition at line 253 of file base_solver.h.

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

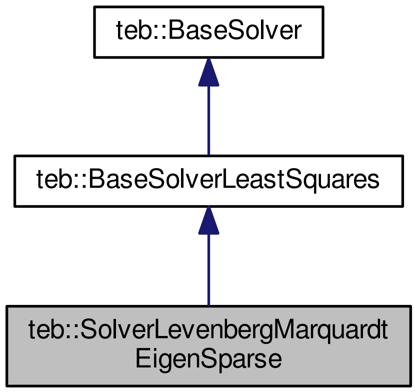
- solver_levenbergmarquardt_eigen_dense.h
- solver_levenbergmarquardt_eigen_dense.cpp

16.33 teb::SolverLevenbergMarquardtEigenSparse Class Reference

Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Sparse matrices version).

`#include <solver_levenbergmarquardt_eigen_sparse.h>`

Inheritance diagram for teb::SolverLevenbergMarquardtEigenSparse:



Public Types

- using VertexContainer = HyperGraph::VertexContainer

    Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).
- using EdgeContainer = HyperGraph::EdgeContainer

    Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).

Public Member Functions

- SolverLevenbergMarquardtEigenSparse ()

    Empty Constructor.
- void setConfig (const Config ∗config)

    Set config including solver settings.
- virtual bool solve (HyperGraph ∗optimizable_graph)

    Solve an optimization problem defined by an hyper-graph.

Public Attributes

- const Config ∗ cfg = nullptr

    Store pointer to Config object.
- EIGEN_MAKE_ALIGNED_OPERATOR_NEW

Protected Member Functions

- virtual bool solveImpl ()

    Solve the actual optimization problem.
- void allocateSparseJacobian ()

    Allocate sparse jacobian.
- void buildJacobian ()

    Construct the jacobian of the complete least-squares optimziation problem.

- void countJacobianColNNZ ()

    Count number of non-zeros per column in order to intialize jacobain sparse matrix.
- virtual void initWorkspaces ()

    Initialize workspaces for the block jacobians and hessians.
- void buildValueVector ()

    Build value / cost vector including all objectives and constraints.
- int getValueDimension () const

    Get dimension of the composed value/cost vector (including objectives and constraints).
- double getChi2 () const

    Calculate the χ^2 error of the optimization problem.
- void adaptWeights ()

    Automatic weight adaptation that increases soft constraint weights after each outer teb iteration.
- void applyIncrement (const Eigen::Ref< const Eigen::VectorXd > &delta)

    Apply a new increment optained by a local optimization to the active vertices.
- void applyOptVec (const Eigen::Ref< const Eigen::VectorXd > &opt_vec)

    Overwrite TEB states and control inputs with new values.
- Eigen::VectorXd getOptVecCopy () const

    Get a copy of the optimization vector (TEB states, control inputs and dt)
- int getOptVecDimension () const

    Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).
- int getDimObjectives () const

    Get dimension of the objective function.
- int getDimEqualities () const

    Get dimension of the equality constraints.
- int getDimInequalities () const

    Get dimension of the inequality constraints.
- void backupVertices ()

    Backup all active vertices.
- void restoreVertices ()

    Restore all values of active vertices from the backup stacks.
- void restoreVerticesButKeepBackup ()

    Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.
- void discardBackupVertices ()

    Discard all values of active vertices from the backup stacks.

Protected Attributes

- Eigen::SparseMatrix< double > _jacobian

    Store the jacobian matrix of the complete least-squares optimziation problem.
- Eigen::VectorXi _nnz_per_col

    Store number of non-zeros per column of the jacobian matrix.
- Eigen::SimplicialLDLT
< Eigen::SparseMatrix< double >
, Eigen::Upper > _sparse_solver

    Eigens sparse solver wrapper.
- Eigen::VectorXd _values

    Store the value vector computed in buildValueVector().
- int _weight_adapt_count = 0

    Store current state of the weight adaptation method adaptWeights().
- double _weight_equalities = 1

    Store current weight for equality soft-constraints.

- double _weight_inequalities = 1

  Store current weight for inequality soft-constraints.
- int _val_dim = -1

  Store dimension of the value vector here (see getValueDimension()).
- VertexContainer ∗ _active_vertices = nullptr

  Pointer to active vertex container (active = non-fixed)
- EdgeContainer ∗ _objectives = nullptr

  Pointer to edges representing objective functions.
- EdgeContainer ∗ _equalities = nullptr

  Pointer to edges representing equality constraints.
- EdgeContainer ∗ _inequalities = nullptr

  Pointer to edges representing inequality constraints.
- int _opt_vec_dim = -1

  Store dimension of the optimziation vector here (see getOptVecDimension()).
- int _objective_dim = -1

  Store dimension of the objective function (see getDimObjectives()).
- int _equalities_dim = -1

  Store dimension of the equality constraints (see getDimEqualities()).
- int _inequalities_dim = -1

  Store dimension of the inequality constraints (see getDimInequalities()).
- unsigned int _no_vert_backups = 0

  Track number of vertices backups made.
- bool _graph_structure_modified = true

  Mark if a new graph structure is available, thus we need to reinitialze all workspaces.

16.33.1 Detailed Description

This solver implements the Levenberg-Marquardt algorithm also implemented in [2]. The underlying linear system is solved using Eigens Sparse Matrix algebra (`http://eigen.tuxfamily.org/dox/group__Topic-SparseSystems.html`).

The jacobian required for the optimization routine is stored in a sparse matrix.

Todo Consider only the upper diagonal part of the hessian?

Author

Christoph Rösmann (`christoph.roesmann@tu-dortmund.de`)

Examples:

integrator_system1.cpp, integrator_system2.cpp, integrator_system_classic_mpc.cpp, integrator_system_mex.cpp, integrator_system_sfun.cpp, linear_system_ode.cpp, linear_system_ode_ctrl_comparison.cpp, linear_system_state_space.cpp, mobile_robot_teb.cpp, rocket_system.cpp, and van_der_pol_system.cpp.

Definition at line 31 of file solver_levenbergmarquardt_eigen_sparse.h.

16.33.2 Member Typedef Documentation

16.33.2.1 using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer `[inherited]`

Definition at line 36 of file base_solver.h.

16.33.2.2 using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer [inherited]

Definition at line 34 of file base_solver.h.

16.33.3 Constructor & Destructor Documentation

16.33.3.1 teb::SolverLevenbergMarquardtEigenSparse::SolverLevenbergMarquardtEigenSparse ( ) [inline]

Definition at line 36 of file solver_levenbergmarquardt_eigen_sparse.h.

16.33.4 Member Function Documentation

16.33.4.1 void teb::BaseSolverLeastSquares::adaptWeights ( ) [protected], [inherited]

This method increases the soft constraint weights for inequalities and equalities after each outer TEB optimization loop (see TebController::optimizeTEB()).

After the outer TEB loop is completed (Config::Teb::teb_iter), the weights are set back to Config::Optim::Solver::Lsq::weight_equalities and Config::Optim::Solver::Lsq::weight_inequalities.

The weights are increased by $\sigma = \sigma_{init} \cdot \gamma^i$. γ denotes the increasement factor Config::Optim::Solver::Lsq::weight_adaptation_factor. i denotes the current iteration number. σ_{init} is obtained from the config (see above).

Call this method at the beginning of solveImpl()!

Remarks

This procedere forces the optimization result to first contract the trajectory in sense of the objective, and afterwards trying to satisfy the constraints. Starting with very high weights in advance could lead to abrupt gradients for the solver. In addition the effect for the objectives (e.g. time optimallity) could be slow, if the corresponding gradients are extremly small in comparision to constraint gradients.

See Also

TebController::optimizeTEB()

Definition at line 100 of file base_solver_least_squares.cpp.

References teb::BaseSolverLeastSquares::_weight_adapt_count, teb::BaseSolverLeastSquares::_weight_equalities, teb::BaseSolverLeastSquares::_weight_inequalities, teb::BaseSolver::cfg, teb::Config::Optim::Solver::lsq, teb::Config::optim, teb::Config::Optim::solver, teb::Config::teb, teb::Config::Teb::teb_iter, teb::Config::Optim::Solver::Lsq::weight_adaptation_factor, teb::Config::Optim::Solver::Lsq::weight_equalities, and teb::Config::Optim::Solver::Lsq::weight_inequalities.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and solveImpl().

16.33.4.2 void teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian ( ) [inline], [protected]

Definition at line 44 of file solver_levenbergmarquardt_eigen_sparse.h.

References _jacobian, _nnz_per_col, teb::BaseSolver::_opt_vec_dim, teb::BaseSolverLeastSquares::_val_dim, and countJacobianColNNZ().

Referenced by buildJacobian().

16.33.4.3 void teb::BaseSolver::applyIncrement (const Eigen::Ref< const Eigen::VectorXd > & *delta*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::plusFree() for each vertex.

Parameters

delta	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.

Definition at line 127 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), solveImpl(), and teb::SolverSQPDense::solveImpl().

16.33.4.4 void teb::BaseSolver::applyOptVec (const Eigen::Ref< const Eigen::VectorXd > & *opt_vec*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::setFree() for each vertex.

Parameters

opt_vec	Eigen::Vector containing all values. The length equals the dimension of all free variables.

Definition at line 144 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

16.33.4.5 void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::push()

Definition at line 223 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), solveImpl(), and teb::SolverSQPDense::solveImpl().

16.33.4.6 void teb::SolverLevenbergMarquardtEigenSparse::buildJacobian ( ) [protected]

This method constructs the jacobian of the complete optimziation problem.

The size of the jacobian is [getValueDimension() x getOptVecDimension()]. This method queries the EdgeType::computeJacobian() of each edge. The column index for each entry is determined using TebVertex::getOptVecIdx and the number of free dimensions for each vertex.

Since this solver relies on soft-constraints for equality and inequality constraints, the chain-rule for calculating derivatives is utilized. Equality constraints are treaten as usual objective, therefore the jacobian does not need to be changed.

For inequality constraints, the derivative of $\max(c(x),0)$ is

$$\begin{cases} \frac{\partial}{\partial x} c_i(x) & \text{if } c_i(x) > 0 \\ 0 & \text{otherwise} \end{cases} = \mathbf{J}_{ineq}(:, j = 1:n) .* \frac{1}{\mathbf{c}(x)} .* \max(\mathbf{c}(x), \mathbf{0})$$

The max operator is applicated component-wise. ' $.*$' corresponds to Matlab's component-wise multiplication.

The resulting jacobian matrix is stored to SolverLevenbergMarquardtEigenSparse::_jacobian;

Todo Check if implemented ordering of column and row iterations fits Eigen's column major representation best.

Definition at line 162 of file solver_levenbergmarquardt_eigen_sparse.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_inequalities, _jacobian, teb::BaseSolver::_objectives, teb::BaseSolverLeastSquares::_weight_equalities, teb::BaseSolverLeastSquares::_weight_inequalities, allocateSparseJacobian(), teb::BaseSolver::cfg, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), teb::VertexType::isFixedComp(), teb::Config::Optim::Solver::lsq, teb::Config::optim, teb::Config::Optim::Solver::Lsq::soft_constr_epsilon, and teb::Config::Optim::solver.

Referenced by solveImpl().

16.33.4.7 void teb::BaseSolverLeastSquares::buildValueVector ( ) [protected], [inherited]

This method constructs the full value/cost vector $\mathbf{f}$ (not $\mathbf{f}^2$) for the underlying least-squares optimization problem.

The lenght of the vector can be obtained using getValueDimension().

Constraints are transformed into costs/objectives using soft constraints:

- Equality constraints $c(x) = 0$.

  These constraints are directly taken as objectives since $c^2(x)$ has a local minima at $x = 0$.

- Inequality constraints $c(x) \leq 0$

  These constraints are transformed using $f = \max(c(x), 0)$ (component-wise).

  This notation is similar to: c(x) <= epsilon ? 0 : c(x).

  Afterwards (in the actual solveImpl() method, the cost function will be squared, that leads to twice differnetiable costs for the constraints, if f is piecewise differentiable once and if it intersectects with the abscissa.

In addition the weights BaseSolverLeastSquares::_weight_equalities and BaseSolverLeastSquares::_weight_inequalities are taken into account in order to weight the "soft constraint objectives". To make the weights comparable to [2] and our Matlab TEB version, we take the square root of both weights, since the least-square problem here is formulated as $f^2(x, \sigma)$ instead of $\sigma f^2(x)$. σ denotes the weight.

The results are stored internally to BaseSolverLeastSquares::_values.

See Also

adaptWeights(), getValueDimension()

Definition at line 31 of file base_solver_least_squares.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_inequalities, teb::BaseSolver::_objectives, teb::BaseSolverLeastSquares::_val_dim, teb::BaseSolverLeastSquares::_values, teb::BaseSolverLeastSquares::_weight_equalities, teb::BaseSolverLeastSquares::_weight_inequalities, teb::BaseSolver::cfg, teb::Config::Optim::Solver::lsq, teb::Config::optim, teb::Config::Optim::Solver::Lsq::soft_constr_epsilon, and teb::Config::Optim::solver.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and solveImpl().

16.33.4.8 void teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ ( ) [protected]

The results are stored to the class member variable SolverLevenbergMarquardtEigenSparse::_nnz_per_col.

Remarks

: Make sure that BaseSolverLeastSquares::_opt_vec_dim is valid!

Definition at line 311 of file solver_levenbergmarquardt_eigen_sparse.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_inequalities, _nnz_per_col, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), and teb::VertexType::isFixedComp().

Referenced by allocateSparseJacobian().

16.33.4.9 void teb::BaseSolver::discardBackupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::discardTop()

Definition at line 229 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), solveImpl(), and teb::SolverSQPDense::solveImpl().

16.33.4.10 double teb::BaseSolverLeastSquares::getChi2 ( ) const [inline], [protected], [inherited]

The χ^2 error is the scalar cost value of the least-squares optimization problem since the total cost function is composed of weighted terms:

$$f(\mathscr{B}) = \mathbf{f}(\mathscr{B})^T \mathbf{f}(\mathscr{B})$$

Weights are included in $\mathbf{f}(\mathscr{B})$. In this case it is $\chi^2 = f(\mathscr{B})$.

Todo Maybe switch to the formulation $f(\mathscr{B}) = \mathbf{f}(\mathscr{B})^T \Omega \mathbf{f}(\mathscr{B})$ similar to g2o and Teb-Matlab.

Definition at line 82 of file base_solver_least_squares.h.

References teb::BaseSolverLeastSquares::_values.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and solveImpl().

16.33.4.11 int teb::BaseSolver::getDimEqualities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file base_solver.h.

References teb::BaseSolver::_equalities.

Referenced by teb::BaseSolver::solve().

16.33.4.12 int teb::BaseSolver::getDimInequalities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file base_solver.h.

References teb::BaseSolver::_inequalities.

Referenced by teb::BaseSolver::solve().

16.33.4.13 int teb::BaseSolver::getDimObjectives ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file base_solver.h.

References teb::BaseSolver::_objectives.

Referenced by teb::BaseSolver::solve().

16.33.4.14 Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::getDataFree() for each vertex.

Remarks

Make sure BaseSolver::_opt_vec_dim is valid (see solve()).

Returns

Eigen::Vector containing all values. The length equals the dimension of all free variables (getOptVecDimension()).

Definition at line 162 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_opt_vec_dim.

16.33.4.15 int teb::BaseSolver::getOptVecDimension ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

See Also

TebController::getActiveVertices(), getValueDimension()

Definition at line 180 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::BaseSolver::solve().

16.33.4.16 int teb::BaseSolverLeastSquares::getValueDimension ( ) const [protected], [inherited]

This method returns the dimension of the full cost/value vector including all objective and constraints.

In particular it sums up all dimensions of single edges stored in the given hyper-graph.

Remarks

Make sure BaseSolver::_objective_dim, BaseSolver::_equalities_dim and BaseSolver::_inequalities_dim are valid (see solve()).

Returns

dimension of the complete value/cost vector.

See Also

buildValueVector(), getOptVecDimension(), getDimObjectives(), getDimEqualities(), getDimInequalities()

Definition at line 73 of file base_solver_least_squares.cpp.

References teb::BaseSolver::_equalities_dim, teb::BaseSolver::_inequalities_dim, and teb::BaseSolver::_objective_dim.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and solveImpl().

16.33.4.17 virtual void teb::BaseSolverLeastSquares::initWorkspaces ( ) [inline], [protected], [virtual], [inherited]

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each EdgeType) or map to existing memory.

Reimplemented from teb::BaseSolver.

Definition at line 43 of file base_solver_least_squares.h.

References teb::BaseSolver::_equalities, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_objectives, PRINT_DEBUG_COND_ONCE, and teb::QUADRATIC.

16.33.4.18 void teb::BaseSolver::restoreVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::pop()

Definition at line 225 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and solveImpl().

16.33.4.19 void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline], [protected], [inherited]

See Also

VertexType::top()

Definition at line 227 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverSQPDense::solveImpl().

16.33.4.20 void teb::BaseSolver::setConfig (const Config * *config*) [inline], [inherited]

Remarks

This function is called from the TEB class within setSolver method.

Parameters

config	Pointer to Config object.

Definition at line 49 of file base_solver.h.

References teb::BaseSolver::cfg.

Referenced by teb::TebController< p, q >::setSolver().

16.33.4.21 virtual bool teb::BaseSolver::solve (HyperGraph * *optimizable_graph*) [inline], [virtual], [inherited]

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with solveImpl().

Todo Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

Parameters

optimizable_graph	pointer to the HyperGraph that should be solved

Definition at line 61 of file base_solver.h.

References teb::BaseSolver::_active_vertices, teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, teb::BaseSolver::_no_vert_backups, teb::BaseSolver::_objective_dim, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::HyperGraph::activeVertices(), teb::BaseSolver::discardBackupVertices(), teb::HyperGraph::equalities(), teb::BaseSolver::getDimEqualities(), teb::BaseSolver::getDimInequalities(), teb::BaseSolver::getDimObjectives(), teb::BaseSolver::getOptVecDimension(), teb::HyperGraph::inequalities(), teb::BaseSolver::initWorkspaces(), teb::HyperGraph::isGraphModified(), teb::HyperGraph::objectives(), and teb::BaseSolver::solveImpl().

16.33.4.22 bool teb::SolverLevenbergMarquardtEigenSparse::solveImpl ( ) [protected], [virtual]

Store results to the vertices of the active vertices container.

Return values

true	optimization was successfull.
false	optimization was not successfull.

Implements teb::BaseSolverLeastSquares.

Definition at line 8 of file solver_levenbergmarquardt_eigen_sparse.cpp.

References _jacobian, teb::BaseSolver::_opt_vec_dim, _sparse_solver, teb::BaseSolverLeastSquares::_val_-dim, teb::BaseSolverLeastSquares::_values, teb::BaseSolverLeastSquares::adaptWeights(), teb::BaseSolver-::applyIncrement(), teb::BaseSolver::backupVertices(), buildJacobian(), teb::BaseSolverLeastSquares::buildValue-Vector(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolverLeastSquares::getChi2(), teb::BaseSolver-LeastSquares::getValueDimension(), and teb::BaseSolver::restoreVertices().

16.33.5 Member Data Documentation

16.33.5.1 VertexContainer* teb::BaseSolver::_active_vertices = nullptr [protected], [inherited]

Definition at line 233 of file base_solver.h.

Referenced by teb::BaseSolver::applyIncrement(), teb::BaseSolver::applyOptVec(), teb::BaseSolver::backup-Vertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::getOptVecCopy(), teb::BaseSolver::get-OptVecDimension(), teb::BaseSolver::restoreVertices(), teb::BaseSolver::restoreVerticesButKeepBackup(), and teb::BaseSolver::solve().

16.33.5.2 EdgeContainer* teb::BaseSolver::_equalities = nullptr [protected], [inherited]

Definition at line 235 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolver-NonlinearProgramDense::buildEqualityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense-::buildJacobian(), buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinear-ProgramDense::calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsEq(), count-JacobianColNNZ(), teb::BaseSolver::getDimEqualities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::-SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and teb::Base-Solver::solve().

16.33.5.3 int teb::BaseSolver::_equalities_dim = -1 [protected], [inherited]

Definition at line 240 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolver-NonlinearProgramDense::buildEqualityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculate-LagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::SolverNloptPackage::get-EqConstrDimFromStorage(), teb::BaseSolverLeastSquares::getValueDimension(), teb::BaseSolverNonlinear-ProgramDense::initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver-::solve(), and teb::SolverSQPDense::solveImpl().

16.33.5.4 bool teb::BaseSolver::_graph_structure_modified = true [protected], [inherited]

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the solve() method.

Definition at line 250 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.33.5.5 EdgeContainer∗ teb::BaseSolver::_inequalities = nullptr [protected], [inherited]

Definition at line 236 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsInEq(), countJacobianColNNZ(), teb::BaseSolver::getDimInequalities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and teb::BaseSolver::solve().

16.33.5.6 int teb::BaseSolver::_inequalities_dim = -1 [protected], [inherited]

Definition at line 241 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::BaseSolverLeastSquares::getValueDimension(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.33.5.7 Eigen::SparseMatrix<double> teb::SolverLevenbergMarquardtEigenSparse::_jacobian [protected]

Definition at line 61 of file solver_levenbergmarquardt_eigen_sparse.h.

Referenced by allocateSparseJacobian(), buildJacobian(), and solveImpl().

16.33.5.8 Eigen::VectorXi teb::SolverLevenbergMarquardtEigenSparse::_nnz_per_col [protected]

Definition at line 63 of file solver_levenbergmarquardt_eigen_sparse.h.

Referenced by allocateSparseJacobian(), and countJacobianColNNZ().

16.33.5.9 unsigned int teb::BaseSolver::_no_vert_backups = 0 [protected], [inherited]

Definition at line 243 of file base_solver.h.

Referenced by teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::restoreVertices(), and teb::BaseSolver::solve().

16.33.5.10 int teb::BaseSolver::_objective_dim = -1 [protected], [inherited]

Definition at line 239 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::getValueDimension(), and teb::BaseSolver::solve().

16.33.5.11 EdgeContainer∗ teb::BaseSolver::_objectives = nullptr [protected], [inherited]

Definition at line 234 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::buildObjectiveValue(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), countJacobianColNNZ(), teb::BaseSolver::getDimObjectives(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), teb::SolverNloptPackage::objectives(), and teb::BaseSolver::solve().

16.33.5.12 int teb::BaseSolver::_opt_vec_dim = -1 [protected], [inherited]

Definition at line 238 of file base_solver.h.

Referenced by allocateSparseJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), countJacobianColNNZ(), teb::BaseSolver::getOptVecCopy(), teb::SolverNloptPackage::getOptVecDimFromStorage(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), solveImpl(), and teb::SolverSQPDense::solveImpl().

16.33.5.13 Eigen::SimplicialLDLT<Eigen::SparseMatrix<double>, Eigen::Upper> teb::SolverLevenbergMarquardtEigenSparse::_sparse_solver [protected]

Check http://eigen.tuxfamily.org/dox/group__TopicSparseSystems.html for further information and different solvers. The second template parameter specifies, whether the upper or lower triangular part should be used. If CHOLMOD was found by CMake, the cholmod supernodal llt version is used rathar than Eigens simplicial ldlt. Define `FORCE_EIGEN_SOLVER` if the default LDLT should sill be used even if CHOLMOD is found.

Definition at line 75 of file solver_levenbergmarquardt_eigen_sparse.h.

Referenced by solveImpl().

16.33.5.14 int teb::BaseSolverLeastSquares::_val_dim = -1 [protected], [inherited]

Definition at line 96 of file base_solver_least_squares.h.

Referenced by allocateSparseJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and solveImpl().

16.33.5.15 Eigen::VectorXd teb::BaseSolverLeastSquares::_values [protected], [inherited]

Definition at line 90 of file base_solver_least_squares.h.

Referenced by teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverLeastSquares::getChi2(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and solveImpl().

16.33.5.16 int teb::BaseSolverLeastSquares::_weight_adapt_count = 0 [protected], [inherited]

Definition at line 92 of file base_solver_least_squares.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights().

16.33.5.17 double teb::BaseSolverLeastSquares::_weight_equalities = 1 [protected], [inherited]

Definition at line 93 of file base_solver_least_squares.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), buildJacobian(), and teb::BaseSolverLeastSquares::buildValueVector().

16.33.5.18 double teb::BaseSolverLeastSquares::_weight_inequalities = 1 [protected], [inherited]

Definition at line 94 of file base_solver_least_squares.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), buildJacobian(), and teb::BaseSolverLeastSquares::buildValueVector().

16.33.5.19 const Config* teb::BaseSolver::cfg = nullptr [inherited]

See Also

setConfig()

Definition at line 95 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::setConfig(), and teb::SolverSQPDense::solveImpl().

16.33.5.20 teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW [inherited]

Definition at line 253 of file base_solver.h.

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

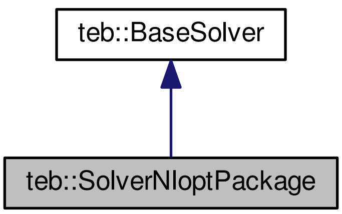
- solver_levenbergmarquardt_eigen_sparse.h
- solver_levenbergmarquardt_eigen_sparse.cpp

16.34 teb::SolverNloptPackage Class Reference

This class wraps the optimization problem to allow solving by NLOPT.

`#include <solver_nlopt_package.h>`

Inheritance diagram for teb::SolverNloptPackage:



Public Types

- using VertexContainer = HyperGraph::VertexContainer

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).

- using EdgeContainer = HyperGraph::EdgeContainer

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).

Public Member Functions

- SolverNloptPackage ()

  Empty Constructor.
- virtual void initWorkspaces ()

  Initialize workspaces for the block jacobians and hessians.
- EdgeContainer ∗ objectives ()

  Return pointer to the edges containing objectives.
- EdgeContainer ∗ constraintsEq ()

  Return pointer to the edges containing equalities.
- EdgeContainer ∗ constraintsInEq ()

  Return pointer to the edges containing inequalities.
- int getOptVecDimFromStorage () const

  Return dimension of the optimization vector (call getOptVecDimension() before)
- int getEqConstrDimFromStorage () const

  Return dimension of the equality constraint vector.
- void seperateInequalitiesAndBounds ()

  Seperate constraintsInEq to bound constraints and general nonlinear inequality constraints and store results internally.
- int getBoundConstrDimFromStorage () const

  Return dimension of the equality constraint vector.
- int getInEqConstrDimFromStorage () const

  Return dimension of the inequality constraint vector.
- void setConfig (const Config ∗config)

  Set config including solver settings.
- virtual bool solve (HyperGraph ∗optimizable_graph)

  Solve an optimization problem defined by an hyper-graph.

Static Public Member Functions

- static double objectiveFunction (unsigned n, const double ∗x, double ∗grad, void ∗solver_object_this)

  Objective function wrapper that iterates all objective edges.
- static void equalityConstraintFunction (unsigned m, double ∗result, unsigned n, const double ∗x, double ∗grad, void ∗solver_object_this)

  Equality constraint function wrapper that iterates all equality constraint edges.
- static void inequalityConstraintFunction (unsigned m, double ∗result, unsigned n, const double ∗x, double ∗grad, void ∗solver_object_this)

  Inequality constraint function wrapper that iterates all inequality constraint edges (excluding bound constraints).

Public Attributes

- const Config ∗ cfg = nullptr

  Store pointer to Config object.
- EIGEN_MAKE_ALIGNED_OPERATOR_NEW

Protected Member Functions

- virtual bool solveImpl ()

  Solve the actual optimization problem.
- void applyIncrement (const Eigen::Ref< const Eigen::VectorXd > &delta)

  Apply a new increment optained by a local optimization to the active vertices.
- void applyOptVec (const Eigen::Ref< const Eigen::VectorXd > &opt_vec)

  Overwrite TEB states and control inputs with new values.
- Eigen::VectorXd getOptVecCopy () const

  Get a copy of the optimization vector (TEB states, control inputs and dt)
- int getOptVecDimension () const

  Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).
- int getDimObjectives () const

  Get dimension of the objective function.
- int getDimEqualities () const

  Get dimension of the equality constraints.
- int getDimInequalities () const

  Get dimension of the inequality constraints.
- void backupVertices ()

  Backup all active vertices.
- void restoreVertices ()

  Restore all values of active vertices from the backup stacks.
- void restoreVerticesButKeepBackup ()

  Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.
- void discardBackupVertices ()

  Discard all values of active vertices from the backup stacks.

Protected Attributes

- EdgeContainer _bound_constraints

  Store bound constraints using seperateInequalitiesAndBounds().
- EdgeContainer _inequalities_without_bounds

  Store general nonlinear constraints using seperateInequalitiesAndBounds().
- int _bound_constraints_dim = 0

  Store dimension of bound constraints using seperateInequalitiesAndBounds().
- int _inequalities_without_bounds_dim = 0

  Store dimension ofgeneral nonlinear constraints using seperateInequalitiesAndBounds().
- Eigen::VectorXd _lower_bounds

  Store lower bounds according to the optimization vector.
- Eigen::VectorXd _upper_bounds

  Store upper bounds according to the optimization vector.
- Eigen::VectorXd _ctol_eq

  Store tolerances for the equality constraints (stopping criteria)
- Eigen::VectorXd _ctol_ineq

  Store tolerances for the inequality constraints (stopping criteria)
- VertexContainer ∗ _active_vertices = nullptr

  Pointer to active vertex container (active = non-fixed)
- EdgeContainer ∗ _objectives = nullptr

  Pointer to edges representing objective functions.
- EdgeContainer ∗ _equalities = nullptr

  Pointer to edges representing equality constraints.

- EdgeContainer ∗ _inequalities = nullptr

    Pointer to edges representing inequality constraints.
- int _opt_vec_dim = -1

    Store dimension of the optimziation vector here (see getOptVecDimension()).
- int _objective_dim = -1

    Store dimension of the objective function (see getDimObjectives()).
- int _equalities_dim = -1

    Store dimension of the equality constraints (see getDimEqualities()).
- int _inequalities_dim = -1

    Store dimension of the inequality constraints (see getDimInequalities()).
- unsigned int _no_vert_backups = 0

    Track number of vertices backups made.
- bool _graph_structure_modified = true

    Mark if a new graph structure is available, thus we need to reinitialze all workspaces.

16.34.1 Detailed Description

Wrapper function for the NLopt optimization library (http://ab-initio.mit.edu/wiki/index.php/-NLopt) The purpose of this solver is not to perform an efficient and fast real-time optimization, rather than having alternative constraint solvers for debugging and testing.

Refer to Config::Optim::Solver::Nlopt for configuration parameters.

Test This class needs more testing (different solvers etc.)

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Definition at line 29 of file solver_nlopt_package.h.

16.34.2 Member Typedef Documentation

16.34.2.1 using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer [inherited]

Definition at line 36 of file base_solver.h.

16.34.2.2 using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer [inherited]

Definition at line 34 of file base_solver.h.

16.34.3 Constructor & Destructor Documentation

16.34.3.1 teb::SolverNloptPackage::SolverNloptPackage ( ) [inline]

Definition at line 34 of file solver_nlopt_package.h.

16.34.4 Member Function Documentation

16.34.4.1 void teb::BaseSolver::applyIncrement (const Eigen::Ref< const Eigen::VectorXd > & *delta*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::plusFree() for each vertex.

Parameters

delta	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.

Definition at line 127 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.34.4.2 void teb::BaseSolver::applyOptVec (const Eigen::Ref< const Eigen::VectorXd > & *opt_vec*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::setFree() for each vertex.

Parameters

opt_vec	Eigen::Vector containing all values. The length equals the dimension of all free variables.

Definition at line 144 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

16.34.4.3 void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::push()

Definition at line 223 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.34.4.4 EdgeContainer* teb::SolverNloptPackage::constraintsEq ( ) [inline]

Definition at line 55 of file solver_nlopt_package.h.

References teb::BaseSolver::_equalities.

16.34.4.5 EdgeContainer* teb::SolverNloptPackage::constraintsInEq ( ) [inline]

Definition at line 56 of file solver_nlopt_package.h.

References teb::BaseSolver::_inequalities.

16.34.4.6 void teb::BaseSolver::discardBackupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::discardTop()

Definition at line 229 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.34.4.7 static void teb::SolverNloptPackage::equalityConstraintFunction (unsigned *m*, double ∗ *result*, unsigned *n*, const double ∗ *x*, double ∗ *grad*, void ∗ *solver_object_this*) [static]

16.34.4.8 int teb::SolverNloptPackage::getBoundConstrDimFromStorage ( ) const [inline]

Definition at line 62 of file solver_nlopt_package.h.

References _bound_constraints_dim.

16.34.4.9 int teb::BaseSolver::getDimEqualities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file base_solver.h.

References teb::BaseSolver::_equalities.

Referenced by teb::BaseSolver::solve().

16.34.4.10 int teb::BaseSolver::getDimInequalities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file base_solver.h.

References teb::BaseSolver::_inequalities.

Referenced by teb::BaseSolver::solve().

16.34.4.11 int teb::BaseSolver::getDimObjectives ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file base_solver.h.

References teb::BaseSolver::_objectives.

Referenced by teb::BaseSolver::solve().

16.34.4.12 int teb::SolverNloptPackage::getEqConstrDimFromStorage ( ) const [inline]

Definition at line 58 of file solver_nlopt_package.h.

References teb::BaseSolver::_equalities_dim.

16.34.4.13 int teb::SolverNloptPackage::getInEqConstrDimFromStorage ( ) const [inline]

Definition at line 63 of file solver_nlopt_package.h.

References _inequalities_without_bounds_dim.

16.34.4.14 Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::getDataFree() for each vertex.

Remarks

Make sure BaseSolver::_opt_vec_dim is valid (see solve()).

Returns

Eigen::Vector containing all values. The length equals the dimension of all free variables (getOptVecDimension()).

Definition at line 162 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_opt_vec_dim.

16.34.4.15 int teb::BaseSolver::getOptVecDimension ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

See Also

TebController::getActiveVertices(), getValueDimension()

Definition at line 180 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::BaseSolver::solve().

16.34.4.16 int teb::SolverNloptPackage::getOptVecDimFromStorage ( ) const [inline]

Definition at line 57 of file solver_nlopt_package.h.

References teb::BaseSolver::_opt_vec_dim.

16.34.4.17 static void teb::SolverNloptPackage::inequalityConstraintFunction (unsigned *m*, double ∗ *result*, unsigned *n*, const double ∗ *x*, double ∗ *grad*, void ∗ *solver_object_this*) [static]

16.34.4.18 virtual void teb::SolverNloptPackage::initWorkspaces ( ) [inline], [virtual]

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each EdgeType) or map to existing memory.

Reimplemented from teb::BaseSolver.

Definition at line 46 of file solver_nlopt_package.h.

References teb::BaseSolver::_equalities, teb::BaseSolver::_inequalities, and teb::BaseSolver::_objectives.

16.34.4.19 static double teb::SolverNloptPackage::objectiveFunction (unsigned *n*, const double ∗ *x*, double ∗ *grad*, void ∗ *solver_object_this*) [static]

16.34.4.20 EdgeContainer∗ teb::SolverNloptPackage::objectives ( ) [inline]

Definition at line 54 of file solver_nlopt_package.h.

References teb::BaseSolver::_objectives.

16.34.4.21 void teb::BaseSolver::restoreVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::pop()

Definition at line 225 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::SolverLevenbergMarquardt-EigenSparse::solveImpl().

16.34.4.22 void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline], [protected], [inherited]

See Also

VertexType::top()

Definition at line 227 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverSQPDense::solveImpl().

16.34.4.23 void teb::SolverNloptPackage::seperateInequalitiesAndBounds ( )

16.34.4.24 void teb::BaseSolver::setConfig (const Config * *config*) [inline], [inherited]

Remarks

This function is called from the TEB class within setSolver method.

Parameters

config	Pointer to Config object.

Definition at line 49 of file base_solver.h.

References teb::BaseSolver::cfg.

Referenced by teb::TebController< p, q >::setSolver().

16.34.4.25 virtual bool teb::BaseSolver::solve (HyperGraph * *optimizable_graph*) [inline], [virtual], [inherited]

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with solveImpl().

Todo Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

Parameters

optimizable_graph	pointer to the HyperGraph that should be solved

Definition at line 61 of file base_solver.h.

References teb::BaseSolver::_active_vertices, teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, teb::BaseSolver::_no_vert_backups, teb::BaseSolver::_objective_dim, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::HyperGraph::activeVertices(), teb::BaseSolver::discardBackupVertices(), teb::HyperGraph::equalities(), teb::BaseSolver::getDimEqualities(), teb::BaseSolver::getDimInequalities(), teb::BaseSolver::getDimObjectives(), teb::BaseSolver::getOptVecDimension(), teb::HyperGraph::inequalities(), teb::BaseSolver::initWorkspaces(), teb::HyperGraph::isGraphModified(), teb::HyperGraph::objectives(), and teb::BaseSolver::solveImpl().

16.34.4.26 virtual bool teb::SolverNloptPackage::solveImpl ( ) [protected], [virtual]

Store results to the vertices of the active vertices container.

Return values

true	optimization was successfull.
false	optimization was not successfull.

Implements teb::BaseSolver.

16.34.5 Member Data Documentation

16.34.5.1 VertexContainer* teb::BaseSolver::_active_vertices = nullptr [protected], [inherited]

Definition at line 233 of file base_solver.h.

Referenced by teb::BaseSolver::applyIncrement(), teb::BaseSolver::applyOptVec(), teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::getOptVecCopy(), teb::BaseSolver::getOptVecDimension(), teb::BaseSolver::restoreVertices(), teb::BaseSolver::restoreVerticesButKeepBackup(), and teb::BaseSolver::solve().

16.34.5.2 EdgeContainer teb::SolverNloptPackage::_bound_constraints [protected]

Definition at line 70 of file solver_nlopt_package.h.

16.34.5.3 int teb::SolverNloptPackage::_bound_constraints_dim = 0 [protected]

Definition at line 72 of file solver_nlopt_package.h.

Referenced by getBoundConstrDimFromStorage().

16.34.5.4 Eigen::VectorXd teb::SolverNloptPackage::_ctol_eq [protected]

Definition at line 77 of file solver_nlopt_package.h.

16.34.5.5 Eigen::VectorXd teb::SolverNloptPackage::_ctol_ineq [protected]

Definition at line 78 of file solver_nlopt_package.h.

16.34.5.6 EdgeContainer* teb::BaseSolver::_equalities = nullptr [protected], [inherited]

Definition at line 235 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), constraintsEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimEqualities(), teb::BaseSolverLeastSquares::initWorkspaces(), initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and teb::BaseSolver::solve().

16.34.5.7 int teb::BaseSolver::_equalities_dim = -1 [protected], [inherited]

Definition at line 240 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolver-NonlinearProgramDense::buildEqualityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculate-LagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), getEqConstrDimFrom-Storage(), teb::BaseSolverLeastSquares::getValueDimension(), teb::BaseSolverNonlinearProgramDense-::initializeLagrangeMultiplier(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), and teb-::SolverSQPDense::solveImpl().

16.34.5.8 bool teb::BaseSolver::_graph_structure_modified = true [protected], [inherited]

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the solve() method.

Definition at line 250 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::init-Workspaces(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.34.5.9 EdgeContainer* teb::BaseSolver::_inequalities = nullptr [protected], [inherited]

Definition at line 236 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolver-NonlinearProgramDense::buildInequalityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense-::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares-::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), constraints-InEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimInequalities(), teb::BaseSolverLeastSquares::initWorkspaces(), initWorkspaces(), teb::BaseSolverNonlinearProgramDense::init-Workspaces(), and teb::BaseSolver::solve().

16.34.5.10 int teb::BaseSolver::_inequalities_dim = -1 [protected], [inherited]

Definition at line 241 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolver-NonlinearProgramDense::buildInequalityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense-::calculateLagrangianHessianNumerically(), teb::SolverSQPDense::calculateMeritDerivative(), teb::BaseSolver-LeastSquares::getValueDimension(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), teb::-SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), and teb::SolverSQPDense::solveImpl().

16.34.5.11 EdgeContainer teb::SolverNloptPackage::_inequalities_without_bounds [protected]

Definition at line 71 of file solver_nlopt_package.h.

16.34.5.12 int teb::SolverNloptPackage::_inequalities_without_bounds_dim = 0 [protected]

Definition at line 73 of file solver_nlopt_package.h.

Referenced by getInEqConstrDimFromStorage().

16.34.5.13 Eigen::VectorXd teb::SolverNloptPackage::_lower_bounds [protected]

Definition at line 75 of file solver_nlopt_package.h.

16.34.5.14 unsigned int teb::BaseSolver::_no_vert_backups = 0 [protected], [inherited]

Definition at line 243 of file base_solver.h.

Referenced by teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::restoreVertices(), and teb::BaseSolver::solve().

16.34.5.15 int teb::BaseSolver::_objective_dim = -1 [protected], [inherited]

Definition at line 239 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::getValueDimension(), and teb::BaseSolver::solve().

16.34.5.16 EdgeContainer∗ teb::BaseSolver::_objectives = nullptr [protected], [inherited]

Definition at line 234 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::buildObjectiveValue(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimObjectives(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), initWorkspaces(), objectives(), and teb::BaseSolver::solve().

16.34.5.17 int teb::BaseSolver::_opt_vec_dim = -1 [protected], [inherited]

Definition at line 238 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getOptVecCopy(), getOptVecDimFromStorage(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

16.34.5.18 Eigen::VectorXd teb::SolverNloptPackage::_upper_bounds [protected]

Definition at line 76 of file solver_nlopt_package.h.

16.34.5.19 const Config∗ teb::BaseSolver::cfg = nullptr [inherited]

See Also

setConfig()

Definition at line 95 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::setConfig(), and teb::SolverSQPDense::solveImpl().

16.34.5.20 teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW [inherited]

Definition at line 253 of file base_solver.h.

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

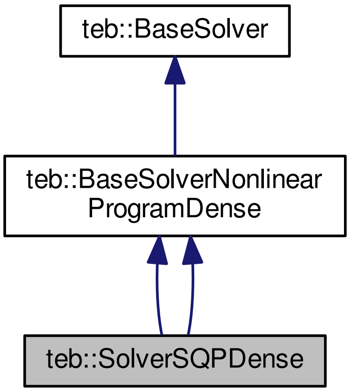
- solver_nlopt_package.h

16.35 teb::SolverSQPDense Class Reference

Dense sequential quadratic programming (SQP) solver for nonlinear programs.

`#include <solver_sqp_dense.h>`

Inheritance diagram for teb::SolverSQPDense:



Public Types

- using VertexContainer = HyperGraph::VertexContainer

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).
- using EdgeContainer = HyperGraph::EdgeContainer

  Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).

Public Member Functions

- SolverSQPDense ()

  Solver Constructor.
- SolverSQPDense ()

  Empty Constructor.
- void setConfig (const Config ∗config)

  Set config including solver settings.
- virtual bool solve (HyperGraph ∗optimizable_graph)

  Solve an optimization problem defined by an hyper-graph.

Public Attributes

- const Config ∗ cfg = nullptr

  Store pointer to Config object.
- EIGEN_MAKE_ALIGNED_OPERATOR_NEW

Protected Types

- using MatMapRowMajor = Eigen::Map< Eigen::Matrix< double,-1,-1, Eigen::RowMajor > >

Protected Member Functions

- virtual void initSolverWorkspace ()
- virtual bool solveImpl ()

  Solve the actual optimization problem.
- bool checkConvergence ()
- double calculateMerit (double alpha) const
- double calculateMeritDerivative (double alpha) const
- virtual void initSolverWorkspace ()
- virtual bool solveImpl ()

  Solve the actual optimization problem.
- double calculateMerit (double rho, double ∗merit_derivative_out=nullptr) const
- virtual void initWorkspaces ()

  Initialize workspaces for the block jacobians and hessians.
- void initializeLagrangeMultiplier ()
- void buildObjectiveValue ()

  Get the sum of all objective values since we solve $f = \min \sum f_k$.
- void buildEqualityConstraintValueVector ()

  Build value / cost vector including all equality constraints.
- void buildInequalityConstraintValueVector ()

  Build value / cost vector including all inequality constraints.
- void buildObjectiveGradient ()
- void buildEqualityConstraintJacobian ()
- void buildInequalityConstraintJacobian ()
- void calculateLagrangianGradient ()
- void calculateLagrangianHessian (Eigen::VectorXd ∗increment=nullptr)
- void calculateLagrangianHessianNumerically ()
- void initHessianBFGS ()
- void calculateLagrangianHessianFullBFGS (Eigen::VectorXd ∗increment)
- void applyIncrement (const Eigen::Ref< const Eigen::VectorXd > &delta)

  Apply a new increment optained by a local optimization to the active vertices.
- void applyOptVec (const Eigen::Ref< const Eigen::VectorXd > &opt_vec)

  Overwrite TEB states and control inputs with new values.
- Eigen::VectorXd getOptVecCopy () const

  Get a copy of the optimization vector (TEB states, control inputs and dt)
- int getOptVecDimension () const

  Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).
- int getDimObjectives () const

  Get dimension of the objective function.
- int getDimEqualities () const

  Get dimension of the equality constraints.
- int getDimInequalities () const

Get dimension of the inequality constraints.
- void backupVertices ()

  Backup all active vertices.
- void restoreVertices ()

  Restore all values of active vertices from the backup stacks.
- void restoreVerticesButKeepBackup ()

  Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.
- void discardBackupVertices ()

  Discard all values of active vertices from the backup stacks.

Protected Attributes

- qpOASES::SQProblem _qsolver
- Eigen::Matrix< double,-1,-1,
  Eigen::RowMajor > _A
- Eigen::VectorXd _lb
- Eigen::VectorXd _ub
- Eigen::VectorXd _delta
- Eigen::VectorXd _qdual
- Eigen::Map< Eigen::VectorXd > _dmultiplier_eq = Eigen::Map<Eigen::VectorXd>(nullptr,0)
- Eigen::Map< Eigen::VectorXd > _dmultiplier_ineq = Eigen::Map<Eigen::VectorXd>(nullptr,0)
- Eigen::VectorXd _merit_grad
- Eigen::VectorXd _increment
- double _merit_alpha = 0
- double _objective_value

  Store the sum of all values computed in buildObjectiveValueVector() since we solve $f = \min \sum f_k$.
- Eigen::VectorXd _equality_values

  Store the value vector computed in buildEqualityConstraintValueVector().
- Eigen::VectorXd _inequality_values

  Store the value vector computed in buildInequalityConstraintValueVector().
- Eigen::VectorXd _objective_gradient

  Store the gradient of the objective value computed in buildObjectiveGradient().
- MatMapRowMajor _equality_jacobian = MatMapRowMajor(nullptr,0,0)

  Store the jacobian matrix of the equality constraint computed in buildEqualityConstraintJacobian().
- MatMapRowMajor _inequality_jacobian = MatMapRowMajor(nullptr,0,0)

  Store the jacobian matrix of the inequality constraint computed in buildInequalityConstraintJacobian().
- Eigen::VectorXd _lagrangian_gradient
- Eigen::VectorXd _lagrangian_gradient_backup

  The gradient from the last step has to be stored for BFGS.
- Eigen::Matrix< double,-1,-1,
  Eigen::RowMajor > _lagrangian_hessian

  Store the hessian of the lagrangian $\nabla^2 L = \nabla^2 (f - \mu^T \mathbf{ceq} - \lambda^T \mathbf{c})$.
- Eigen::VectorXd _multiplier_ineq
- Eigen::VectorXd _multiplier_eq
- VertexContainer ∗ _active_vertices = nullptr

  Pointer to active vertex container (active = non-fixed)
- EdgeContainer ∗ _objectives = nullptr

  Pointer to edges representing objective functions.
- EdgeContainer ∗ _equalities = nullptr

  Pointer to edges representing equality constraints.
- EdgeContainer ∗ _inequalities = nullptr

  Pointer to edges representing inequality constraints.

- int _opt_vec_dim = -1

  Store dimension of the optimziation vector here (see getOptVecDimension()).
- int _objective_dim = -1

  Store dimension of the objective function (see getDimObjectives()).
- int _equalities_dim = -1

  Store dimension of the equality constraints (see getDimEqualities()).
- int _inequalities_dim = -1

  Store dimension of the inequality constraints (see getDimInequalities()).
- unsigned int _no_vert_backups = 0

  Track number of vertices backups made.
- bool _graph_structure_modified = true

  Mark if a new graph structure is available, thus we need to reinitialze all workspaces.

16.35.1 Detailed Description

Todo The documentation of this class, after the SQP sovler passed a couple of remaining (robustness) tests

Author

Christoph Rösmann (`christoph.roesmann@tu-dortmund.de`)

Definition at line 24 of file solver_sqp_dense.h.

16.35.2 Member Typedef Documentation

16.35.2.1 using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer `[inherited]`

Definition at line 36 of file base_solver.h.

16.35.2.2 using teb::BaseSolverNonlinearProgramDense::MatMapRowMajor = Eigen::Map<Eigen::Matrix<double,-1,-1,Eigen::RowMajor>> `[protected]`, `[inherited]`

Definition at line 110 of file base_solver_nonlinear_program_dense.h.

16.35.2.3 using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer `[inherited]`

Definition at line 34 of file base_solver.h.

16.35.3 Constructor & Destructor Documentation

16.35.3.1 teb::SolverSQPDense::SolverSQPDense ( ) `[inline]`

Definition at line 29 of file solver_sqp_dense.h.

References _qsolver.

16.35.3.2 teb::SolverSQPDense::SolverSQPDense ( ) `[inline]`

Definition at line 19 of file solver_sqp_dense_backup.h.

16.35.4 Member Function Documentation

16.35.4.1 void teb::BaseSolver::applyIncrement (const Eigen::Ref< const Eigen::VectorXd > & *delta*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::plusFree() for each vertex.

Parameters

delta	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.

Definition at line 127 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and solveImpl().

16.35.4.2 void teb::BaseSolver::applyOptVec (const Eigen::Ref< const Eigen::VectorXd > & *opt_vec*) [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::setFree() for each vertex.

Parameters

opt_vec	Eigen::Vector containing all values. The length equals the dimension of all free variables.

Definition at line 144 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

16.35.4.3 void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::push()

Definition at line 223 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and solveImpl().

16.35.4.4 void teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian ( ) [protected], [inherited]

Definition at line 154 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, teb::BaseSolverNonlinearProgramDense::_equality_jacobian, teb::BaseSolverNonlinearProgramDense::_equality_values, teb::BaseSolver::_opt_vec_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), and teb::VertexType::isFixedComp().

Referenced by solveImpl().

16.35.4.5 void teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector ( ) [protected], [inherited]

Definition at line 30 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, and teb::BaseSolverNonlinearProgramDense::_equality_values.

Referenced by solveImpl().

16.35.4.6 void teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian ( ) [protected], [inherited]

Definition at line 200 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, teb::BaseSolverNonlinearProgram-Dense::_inequality_jacobian, teb::BaseSolverNonlinearProgramDense::_inequality_values, teb::BaseSolver::_opt-_vec_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), and teb::-VertexType::isFixedComp().

Referenced by solveImpl().

16.35.4.7 void teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector ( ) [protected], [inherited]

Definition at line 54 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, and teb::BaseSolverNonlinear-ProgramDense::_inequality_values.

Referenced by solveImpl().

16.35.4.8 void teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient ( ) [protected], [inherited]

Definition at line 82 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolverNonlinearProgramDense::_objective_gradient, teb::BaseSolver::_objectives, teb::-BaseSolver::_opt_vec_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::is-FixedAny(), teb::VertexType::isFixedComp(), teb::LINEAR_SQUARED, and teb::NONLINEAR_SQUARED.

Referenced by solveImpl().

16.35.4.9 void teb::BaseSolverNonlinearProgramDense::buildObjectiveValue ( ) [protected], [inherited]

Definition at line 7 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolverNonlinearProgramDense::_objective_value, teb::BaseSolver::_objectives, teb::LINEA-R_SQUARED, and teb::NONLINEAR_SQUARED.

Referenced by solveImpl().

16.35.4.10 void teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient ( ) [inline], [protected], [inherited]

Definition at line 98 of file base_solver_nonlinear_program_dense.h.

References teb::BaseSolverNonlinearProgramDense::_equality_jacobian, teb::BaseSolverNonlinearProgram-Dense::_inequality_jacobian, teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient, teb::BaseSolver-NonlinearProgramDense::_multiplier_eq, teb::BaseSolverNonlinearProgramDense::_multiplier_ineq, and teb::-BaseSolverNonlinearProgramDense::_objective_gradient.

Referenced by solveImpl().

16.35.4.11 void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian (Eigen::VectorXd * *increment* = nullptr) [protected], [inherited]

Definition at line 245 of file base_solver_nonlinear_program_dense.cpp.

References teb::BLOCK_BFGS, teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBF-GS(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::BaseSolver::cfg, teb::FULL_BFGS, teb::FULL_BFGS_WITH_STRUCTURE_FILTER, teb::Config::Optim::Solver::NonlinearProgram::hessian, teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_method, teb::Config::Optim::Solver::nonlin_prog, teb::NUMERIC, teb::Config::optim, PRINT_ERROR, PRINT_INFO, teb::Config::Optim::solver, and teb::ZERO_HESSIAN.

Referenced by solveImpl().

16.35.4.12 void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS (Eigen::VectorXd * *increment*) [protected], [inherited]

Definition at line 520 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient, teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient_backup, teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian, teb::Config::Optim::Solver::NonlinearProgram::Hessian::bfgs_damped_mode, teb::BaseSolver::cfg, teb::Config::Optim::Solver::NonlinearProgram::hessian, teb::Config::Optim::Solver::nonlin_prog, teb::Config::optim, PRINT_DEBUG, and teb::Config::Optim::solver.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian().

16.35.4.13 void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically ( ) [protected], [inherited]

Definition at line 273 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, teb::BaseSolverNonlinearProgramDense::_equality_values, teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, teb::BaseSolverNonlinearProgramDense::_inequality_values, teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian, teb::BaseSolverNonlinearProgramDense::_multiplier_eq, teb::BaseSolverNonlinearProgramDense::_multiplier_ineq, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), teb::VertexType::isFixedComp(), teb::LINEAR, teb::LINEAR_SQUARED, teb::NONLINEAR_ONCE_DIFF, and teb::NONLINEAR_SQUARED.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), and teb::BaseSolverNonlinearProgramDense::initHessianBFGS().

16.35.4.14 double teb::SolverSQPDense::calculateMerit (double *rho,* double * *merit_derivative_out =* nullptr) const [inline], [protected]

Definition at line 205 of file solver_sqp_dense_backup.h.

References _delta, teb::BaseSolverNonlinearProgramDense::_equality_values, teb::BaseSolverNonlinearProgramDense::_inequality_values, teb::BaseSolverNonlinearProgramDense::_objective_gradient, and teb::BaseSolverNonlinearProgramDense::_objective_value.

16.35.4.15 double teb::SolverSQPDense::calculateMerit (double *alpha*) const [inline], [protected]

Definition at line 234 of file solver_sqp_dense.h.

References teb::BaseSolverNonlinearProgramDense::_equality_values, teb::BaseSolverNonlinearProgramDense::_inequality_values, and teb::BaseSolverNonlinearProgramDense::_objective_value.

Referenced by solveImpl().

16.35.4.16 double teb::SolverSQPDense::calculateMeritDerivative (double *alpha*) const [inline], [protected]

Definition at line 243 of file solver_sqp_dense.h.

References _delta, teb::BaseSolver::_equalities_dim, teb::BaseSolverNonlinearProgramDense::_equality_jacobian, teb::BaseSolverNonlinearProgramDense::_equality_values, teb::BaseSolver::_inequalities_dim, teb::BaseSolverNonlinearProgramDense::_inequality_jacobian, teb::BaseSolverNonlinearProgramDense::_inequality_values, and teb::BaseSolverNonlinearProgramDense::_objective_gradient.

Referenced by solveImpl().

16.35.4.17 bool teb::SolverSQPDense::checkConvergence ( ) [inline], [protected]

Definition at line 228 of file solver_sqp_dense.h.

References teb::BaseSolverNonlinearProgramDense::_equality_values, and teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient.

Referenced by solveImpl().

16.35.4.18 void teb::BaseSolver::discardBackupVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::discardTop()

Definition at line 229 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and solveImpl().

16.35.4.19 int teb::BaseSolver::getDimEqualities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file base_solver.h.

References teb::BaseSolver::_equalities.

Referenced by teb::BaseSolver::solve().

16.35.4.20 int teb::BaseSolver::getDimInequalities ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file base_solver.h.

References teb::BaseSolver::_inequalities.

Referenced by teb::BaseSolver::solve().

16.35.4.21 int teb::BaseSolver::getDimObjectives ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file base_solver.h.

References teb::BaseSolver::_objectives.

Referenced by teb::BaseSolver::solve().

16.35.4.22 Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const [inline], [protected], [inherited]

This method iterates the active vertices container and calls VertexType::getDataFree() for each vertex.

Remarks

Make sure BaseSolver::_opt_vec_dim is valid (see solve()).

Returns

Eigen::Vector containing all values. The length equals the dimension of all free variables (getOptVecDimension()).

Definition at line 162 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_opt_vec_dim.

16.35.4.23 int teb::BaseSolver::getOptVecDimension ( ) const [inline], [protected], [inherited]

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

See Also

TebController::getActiveVertices(), getValueDimension()

Definition at line 180 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by teb::BaseSolver::solve().

16.35.4.24 void teb::BaseSolverNonlinearProgramDense::initHessianBFGS ( ) [protected], [inherited]

Definition at line 489 of file base_solver_nonlinear_program_dense.cpp.

References teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient, teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient_backup, teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian, teb::BaseSolver::_opt_vec_dim, teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::BaseSolver::cfg, teb::Config::Optim::Solver::NonlinearProgram::hessian, teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init, teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init_identity_scale, teb::IDENTITY, teb::Config::Optim::Solver::nonlin_prog, teb::NUMERIC, teb::Config::optim, PRINT_ERROR, teb::Config::Optim::solver, and teb::ZERO.

Referenced by solveImpl().

16.35.4.25 void teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier ( ) [inline], [protected], [inherited]

Definition at line 70 of file base_solver_nonlinear_program_dense.h.

References teb::BaseSolver::_equalities_dim, teb::BaseSolver::_inequalities_dim, teb::BaseSolverNonlinearProgramDense::_multiplier_eq, and teb::BaseSolverNonlinearProgramDense::_multiplier_ineq.

Referenced by solveImpl().

16.35.4.26 virtual void teb::SolverSQPDense::initSolverWorkspace ( ) [inline], [protected], [virtual]

Implements teb::BaseSolverNonlinearProgramDense.

Definition at line 24 of file solver_sqp_dense_backup.h.

References _A, _delta, _dmultiplier_eq, _dmultiplier_ineq, teb::BaseSolver::_equalities_dim, teb::BaseSolverNonlinearProgramDense::_equality_jacobian, teb::BaseSolverNonlinearProgramDense::_equality_values, teb::BaseSolver::_inequalities_dim, teb::BaseSolverNonlinearProgramDense::_inequality_jacobian, teb::BaseSolverNonlinearProgramDense::_inequality_values, _lb, _merit_grad, teb::BaseSolver::_opt_vec_dim, _qdual, _ub, and INF.

16.35.4.27 virtual void teb::SolverSQPDense::initSolverWorkspace ( ) [inline], [protected], [virtual]

Implements teb::BaseSolverNonlinearProgramDense.

Definition at line 37 of file solver_sqp_dense.h.

References _A, _delta, _dmultiplier_eq, _dmultiplier_ineq, teb::BaseSolver::_equalities_dim, teb::BaseSolverNonlinearProgramDense::_equality_jacobian, teb::BaseSolverNonlinearProgramDense::_equality_values, teb::BaseSolver::_inequalities_dim, teb::BaseSolverNonlinearProgramDense::_inequality_jacobian, teb::BaseSolverNonlinearProgramDense::_inequality_values, _lb, _merit_alpha, _merit_grad, teb::BaseSolver::_opt_vec_dim, _qdual, _qsolver, _ub, teb::Config::Optim::Solver::NonlinearProgram::LineSearch::alpha_init, teb::BaseSolver::cfg, teb::Config::Optim::Solver::NonlinearProgram::hessian, teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_method, INF, teb::Config::Optim::Solver::NonlinearProgram::linesearch, teb::Config::Optim::Solver::nonlin_prog, teb::Config::optim, teb::Config::Optim::solver, and teb::ZERO_HESSIAN.

Referenced by solveImpl().

16.35.4.28 virtual void teb::BaseSolverNonlinearProgramDense::initWorkspaces ( ) [inline], [protected], [virtual], [inherited]

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each EdgeType) or map to existing memory.

Reimplemented from teb::BaseSolver.

Definition at line 46 of file base_solver_nonlinear_program_dense.h.

References teb::BaseSolver::_equalities, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_objectives, teb::LINEAR_SQUARED, teb::NONLINEAR_SQUARED, and PRINT_DEBUG_COND_ONCE.

16.35.4.29 void teb::BaseSolver::restoreVertices ( ) [inline], [protected], [inherited]

See Also

VertexType::pop()

Definition at line 225 of file base_solver.h.

References teb::BaseSolver::_active_vertices, and teb::BaseSolver::_no_vert_backups.

Referenced by teb::SolverLevenbergMarquardtEigenDense::solveImpl(), and teb::SolverLevenbergMarquardtEigenSparse::solveImpl().

16.35.4.30 void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline], [protected], [inherited]

See Also

VertexType::top()

Definition at line 227 of file base_solver.h.

References teb::BaseSolver::_active_vertices.

Referenced by solveImpl().

16.35.4.31 void teb::BaseSolver::setConfig (const Config ∗ *config*) [inline], [inherited]

Remarks

This function is called from the TEB class within setSolver method.

Parameters

config	Pointer to Config object.

Definition at line 49 of file base_solver.h.

References teb::BaseSolver::cfg.

Referenced by teb::TebController< p, q >::setSolver().

16.35.4.32 virtual bool teb::BaseSolver::solve (**HyperGraph** * *optimizable_graph*) [inline], [virtual], [inherited]

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with solveImpl().

Todo Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

Parameters

optimizable_-graph	pointer to the HyperGraph that should be solved

Definition at line 61 of file base_solver.h.

References teb::BaseSolver::_active_vertices, teb::BaseSolver::_equalities, teb::BaseSolver::_equalities_dim, teb::BaseSolver::_graph_structure_modified, teb::BaseSolver::_inequalities, teb::BaseSolver::_inequalities_dim, teb::BaseSolver::_no_vert_backups, teb::BaseSolver::_objective_dim, teb::BaseSolver::_objectives, teb::BaseSolver::_opt_vec_dim, teb::HyperGraph::activeVertices(), teb::BaseSolver::discardBackupVertices(), teb::HyperGraph::equalities(), teb::BaseSolver::getDimEqualities(), teb::BaseSolver::getDimInequalities(), teb::BaseSolver::getDimObjectives(), teb::BaseSolver::getOptVecDimension(), teb::HyperGraph::inequalities(), teb::BaseSolver::initWorkspaces(), teb::HyperGraph::isGraphModified(), teb::HyperGraph::objectives(), and teb::BaseSolver::solveImpl().

16.35.4.33 virtual bool teb::SolverSQPDense::solveImpl ( ) [inline], [protected], [virtual]

Store results to the vertices of the active vertices container.

Return values

true	optimization was successfull.
false	optimization was not successfull.

Implements teb::BaseSolverNonlinearProgramDense.

Definition at line 63 of file solver_sqp_dense_backup.h.

References _A, _delta, _dmultiplier_eq, _dmultiplier_ineq, teb::BaseSolver::_equalities_dim, teb::BaseSolverNonlinearProgramDense::_equality_values, teb::BaseSolver::_inequalities_dim, teb::BaseSolverNonlinearProgramDense::_inequality_values, teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian, _lb, teb::BaseSolverNonlinearProgramDense::_multiplier_eq, teb::BaseSolverNonlinearProgramDense::_multiplier_ineq, teb::BaseSolverNonlinearProgramDense::_objective_gradient, teb::BaseSolver::_opt_vec_dim, _qdual, _ub, teb::BaseSolver::applyIncrement(), teb::BaseSolver::backupVertices(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::buildObjectiveValue(), calculateMerit(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), initSolverWorkspace(), PRINT_DEBUG, and teb::BaseSolver::restoreVerticesButKeepBackup().

16.35.4.34 virtual bool teb::SolverSQPDense::solveImpl ( ) [inline], [protected], [virtual]

Store results to the vertices of the active vertices container.

Return values

true	optimization was successfull.
false	optimization was not successfull.

Implements teb::BaseSolverNonlinearProgramDense.

Definition at line 86 of file solver_sqp_dense.h.

References _A, _delta, _dmultiplier_eq, _dmultiplier_ineq, teb::BaseSolver::_equalities_dim, teb::BaseSolverNonlinearProgramDense::_equality_values, teb::BaseSolver::_graph_structure_modified, _increment, teb::BaseSolver::_inequalities_dim, teb::BaseSolverNonlinearProgramDense::_inequality_values, teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian, _lb, _merit_alpha, teb::BaseSolverNonlinearProgramDense::_multiplier_eq, teb::BaseSolverNonlinearProgramDense::_multiplier_ineq, teb::BaseSolverNonlinearProgramDense::_objective_gradient, _qdual, _qsolver, _ub, teb::BaseSolver::applyIncrement(), teb::BaseSolver::backupVertices(), teb::Config::Optim::Solver::NonlinearProgram::LineSearch::beta, teb::BLOCK_BFGS, teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::buildObjectiveValue(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), calculateMerit(), calculateMeritDerivative(), teb::BaseSolver::cfg, checkConvergence(), teb::BaseSolver::discardBackupVertices(), teb::FULL_BFGS, teb::FULL_BFGS_WITH_STRUCTURE_FILTER, teb::Config::Optim::Solver::NonlinearProgram::hessian, teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_method, teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), initSolverWorkspace(), teb::Config::Optim::Solver::NonlinearProgram::linesearch, teb::Config::Optim::Solver::nonlin_prog, teb::Config::optim, PRINT_DEBUG, PRINT_DEBUG_ONCE, teb::BaseSolver::restoreVerticesButKeepBackup(), teb::Config::Optim::Solver::NonlinearProgram::LineSearch::sigma, and teb::Config::Optim::solver.

16.35.5 Member Data Documentation

16.35.5.1 Eigen::Matrix< double,-1,-1, Eigen::RowMajor > teb::SolverSQPDense::_A [protected]

Definition at line 284 of file solver_sqp_dense.h.

Referenced by initSolverWorkspace(), and solveImpl().

16.35.5.2 VertexContainer* teb::BaseSolver::_active_vertices = nullptr [protected], [inherited]

Definition at line 233 of file base_solver.h.

Referenced by teb::BaseSolver::applyIncrement(), teb::BaseSolver::applyOptVec(), teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::getOptVecCopy(), teb::BaseSolver::getOptVecDimension(), teb::BaseSolver::restoreVertices(), teb::BaseSolver::restoreVerticesButKeepBackup(), and teb::BaseSolver::solve().

16.35.5.3 Eigen::VectorXd teb::SolverSQPDense::_delta [protected]

Definition at line 288 of file solver_sqp_dense.h.

Referenced by calculateMerit(), calculateMeritDerivative(), initSolverWorkspace(), and solveImpl().

16.35.5.4 Eigen::Map< Eigen::VectorXd > teb::SolverSQPDense::_dmultiplier_eq = Eigen::Map<Eigen::VectorXd>(nullptr,0) [protected]

Definition at line 290 of file solver_sqp_dense.h.

Referenced by initSolverWorkspace(), and solveImpl().

16.35.5.5 Eigen::Map< Eigen::VectorXd > teb::SolverSQPDense::_dmultiplier_ineq = Eigen::Map<Eigen::VectorXd>(nullptr,0) [protected]

Definition at line 291 of file solver_sqp_dense.h.

Referenced by initSolverWorkspace(), and solveImpl().

16.35.5.6 EdgeContainer* teb::BaseSolver::_equalities = nullptr [protected], [inherited]

Definition at line 235 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimEqualities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and teb::BaseSolver::solve().

16.35.5.7 int teb::BaseSolver::_equalities_dim = -1 [protected], [inherited]

Definition at line 240 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), calculateMeritDerivative(), teb::SolverNloptPackage::getEqConstrDimFromStorage(), teb::BaseSolverLeastSquares::getValueDimension(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), initSolverWorkspace(), teb::BaseSolver::solve(), and solveImpl().

16.35.5.8 MatMapRowMajor teb::BaseSolverNonlinearProgramDense::_equality_jacobian = MatMapRowMajor(nullptr,0,0) [protected], [inherited]

Definition at line 117 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient(), calculateMeritDerivative(), and initSolverWorkspace().

16.35.5.9 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_equality_values [protected], [inherited]

Definition at line 113 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), calculateMerit(), calculateMeritDerivative(), checkConvergence(), initSolverWorkspace(), and solveImpl().

16.35.5.10 bool teb::BaseSolver::_graph_structure_modified = true [protected], [inherited]

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the solve() method.

Definition at line 250 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), teb::BaseSolver::solve(), and solveImpl().

16.35.5.11 Eigen::VectorXd teb::SolverSQPDense::_increment [protected]

Definition at line 294 of file solver_sqp_dense.h.

Referenced by solveImpl().

16.35.5.12 EdgeContainer* teb::BaseSolver::_inequalities = nullptr [protected], [inherited]

Definition at line 236 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverNloptPackage::constraintsInEq(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimInequalities(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), and teb::BaseSolver::solve().

16.35.5.13 int teb::BaseSolver::_inequalities_dim = -1 [protected], [inherited]

Definition at line 241 of file base_solver.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), calculateMeritDerivative(), teb::BaseSolverLeastSquares::getValueDimension(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), initSolverWorkspace(), teb::BaseSolver::solve(), and solveImpl().

16.35.5.14 MatMapRowMajor teb::BaseSolverNonlinearProgramDense::_inequality_jacobian = MatMapRowMajor(nullptr,0,0) [protected], [inherited]

Definition at line 118 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient(), calculateMeritDerivative(), and initSolverWorkspace().

16.35.5.15 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_inequality_values [protected], [inherited]

Definition at line 114 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), calculateMerit(), calculateMeritDerivative(), initSolverWorkspace(), and solveImpl().

16.35.5.16 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient [protected], [inherited]

Definition at line 120 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), checkConvergence(), and teb::BaseSolverNonlinearProgramDense::initHessianBFGS().

16.35.5.17 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient_backup [protected], [inherited]

Definition at line 121 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), and teb::BaseSolverNonlinearProgramDense::initHessianBFGS().

16.35.5.18 Eigen::Matrix<double,-1,-1,Eigen::RowMajor> teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian [protected], [inherited]

Definition at line 122 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), and solveImpl().

16.35.5.19 Eigen::VectorXd teb::SolverSQPDense::_lb [protected]

Definition at line 285 of file solver_sqp_dense.h.

Referenced by initSolverWorkspace(), and solveImpl().

16.35.5.20 double teb::SolverSQPDense::_merit_alpha = 0 [protected]

Definition at line 296 of file solver_sqp_dense.h.

Referenced by initSolverWorkspace(), and solveImpl().

16.35.5.21 Eigen::VectorXd teb::SolverSQPDense::_merit_grad [protected]

Definition at line 292 of file solver_sqp_dense.h.

Referenced by initSolverWorkspace().

16.35.5.22 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_multiplier_eq [protected], [inherited]

Definition at line 125 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), and solveImpl().

16.35.5.23 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_multiplier_ineq [protected], [inherited]

Definition at line 124 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier(), and solveImpl().

16.35.5.24 unsigned int teb::BaseSolver::_no_vert_backups = 0 [protected], [inherited]

Definition at line 243 of file base_solver.h.

Referenced by teb::BaseSolver::backupVertices(), teb::BaseSolver::discardBackupVertices(), teb::BaseSolver::restoreVertices(), and teb::BaseSolver::solve().

16.35.5.25 int teb::BaseSolver::_objective_dim = -1 [protected], [inherited]

Definition at line 239 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::getValueDimension(), and teb::BaseSolver::solve().

16.35.5.26 Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_objective_gradient [protected], [inherited]

Definition at line 116 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient(), calculateMerit(), calculateMeritDerivative(), and solveImpl().

16.35.5.27 double teb::BaseSolverNonlinearProgramDense::_objective_value [protected], [inherited]

Definition at line 112 of file base_solver_nonlinear_program_dense.h.

Referenced by teb::BaseSolverNonlinearProgramDense::buildObjectiveValue(), and calculateMerit().

16.35.5.28 EdgeContainer* teb::BaseSolver::_objectives = nullptr [protected], [inherited]

Definition at line 234 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::buildObjectiveValue(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getDimObjectives(), teb::BaseSolverLeastSquares::initWorkspaces(), teb::SolverNloptPackage::initWorkspaces(), teb::BaseSolverNonlinearProgramDense::initWorkspaces(), teb::SolverNloptPackage::objectives(), and teb::BaseSolver::solve().

16.35.5.29 int teb::BaseSolver::_opt_vec_dim = -1 [protected], [inherited]

Definition at line 238 of file base_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian(), teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getOptVecCopy(), teb::SolverNloptPackage::getOptVecDimFromStorage(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), initSolverWorkspace(), teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and solveImpl().

16.35.5.30 Eigen::VectorXd teb::SolverSQPDense::_qdual [protected]

Definition at line 289 of file solver_sqp_dense.h.

Referenced by initSolverWorkspace(), and solveImpl().

16.35.5.31 qpOASES::SQProblem teb::SolverSQPDense::_qsolver [protected]

Definition at line 282 of file solver_sqp_dense.h.

Referenced by initSolverWorkspace(), solveImpl(), and SolverSQPDense().

16.35.5.32 Eigen::VectorXd teb::SolverSQPDense::_ub [protected]

Definition at line 286 of file solver_sqp_dense.h.

Referenced by initSolverWorkspace(), and solveImpl().

16.35.5.33 const Config* teb::BaseSolver::cfg = nullptr [inherited]

See Also

setConfig()

Definition at line 95 of file base_solver.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::initHessianBFGS(), initSolverWorkspace(), teb::BaseSolver::setConfig(), and solveImpl().

16.35.5.34 teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW [inherited]

Definition at line 253 of file base_solver.h.

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

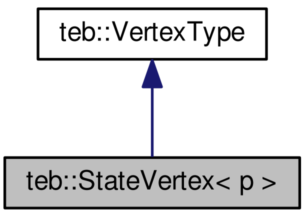
- solver_sqp_dense.h
- solver_sqp_dense_backup.h

16.36 teb::StateVertex< p > Class Template Reference

Vertex that contains the StateVector.

`#include <teb_vertices.h>`

Inheritance diagram for teb::StateVertex< p >:



Public Types

- using StateVector = Eigen::Matrix< double, p, 1 >

  Typedef for state vector with p states [p x 1].
- using ScalarType = Eigen::Matrix< double, 1, 1 >

Typedef for a 1D-scalar value represented as Eigen::Matrix type [1 x 1].

- using FixedStates = Eigen::Matrix< bool, p, 1 >

  Typedef for logical mask that stores fixed and unfixed states [p x 1].

- using BackupStackType = std::stack< StateVector, std::vector< StateVector, Eigen::aligned_allocator< StateVector > > >

  Typdef for the backup stack.

- using VertexContainer = std::vector< VertexType ∗ >

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).

Public Member Functions

- StateVertex ()

  Default constructor: states vector initialized to zero.

- StateVertex (const Eigen::Ref< const StateVector > &states)

  Construct StateVertex with predefined state values.

- StateVertex (const double ∗states)

  Construct StateVertex with predefined state values and predefined control inputs.

- StateVertex (const StateVertex< p > &obj)

  Copy constructor.

- virtual int dimension () const

  Return number of elements/values/components stored in this vertex.

- virtual int dimensionFree () const

  Return only the dimension of free (unfixed) variables.

- virtual void plus (const VertexType ∗rhs)

  Define the increment for the vertex: x = x + rhs.

- virtual void plus (const double ∗rhs)

  Define the increment for the vertex: x = x + rhs (rhs[] with dimension()=p+q values)

- virtual void plusFree (const double ∗rhs)

  Define the increment for the vertex: x = x + rhs (But only add FREE variables, rhs[] with dimensionFree() values).

- virtual void setFree (const double ∗rhs)

  Overwrite all free variables with the values given by rhs (rhs[] with dimensionFree() values).

- virtual void getDataFree (double ∗target_vec) const

  Return a copy of the free (unfixed) values as a single array (length: dimensionFree()).

- virtual const double & getData (unsigned int idx) const

  Return pointer to data with the index `idx`).

- const StateVector & states () const

  Get StateVector (read-only).

- StateVector & states ()

  Get StateVector.

- virtual void setStates (const Eigen::Ref< const StateVector > &s)

  Set or update StateVector.

- virtual void setStates (double state)

  Set or update only first component of the StateVector (useful for 1d states).

- const FixedStates & fixed_states () const

  Get logical map that defines fixed and unfixed states (fixed = `true`).

- void setFixedAll (bool fixed)

  Set all states of this object fixed or unfixed.

- void setFixedState (unsigned int state_idx, bool fixed)

  Set a single component of the StateVector to fixed/unfixed.

- void setFixedStates (bool fixed)

Set all states of the StateVector to fixed/unfixed.

- void setFixedStates (const bool ∗fixed)

  Set selected states of the StateVector to fixed/unfixed at once.
- void setFixedStates (const Eigen::Ref< const FixedStates > &fixed_states)

  Set selected states of the StateVector to fixed/unfixed at once (Eigen version).
- virtual bool isFixedAll () const

  return `true`, if ALL values of this vertex are fixed and therefore are not necessary for optimization.
- bool isFixedAny () const

  return `true`, if ANY element/value/component of this vertex is fixed and therefore the vertex is relevant for optimization.
- virtual bool isFixedComp (int idx) const

  Check if selected element of the StateVector is fixed.
- virtual void push ()

  This function should store all values into a internal backup stack.
- virtual void pop ()

  This function should restore the previously stored values of the backup stack and remove them from the stack.
- virtual void top ()

  This function should restore the previously stored values of the backup stack WITHOUT removing them from the stack.
- virtual void discardTop ()

  This function should delete the previously made backup from the stack without restoring it.
- virtual unsigned int stackSize () const

  This function should return the current size/number of backups of the backup stack.
- StateVertex< p > & operator= (const StateVertex< p > &rhs)
- void setOptVecIdx (int opt_vec_idx)

  Update the position of the first value of this vertex inside the optimization vector / Hessian.
- int getOptVecIdx () const

  Get the position of the first value of this Vertex inside the optimization vector / Hessian.

Static Public Attributes

- static const int DimStates = p

  Number of states / Size of the state vector as static variable.
- static const int Dimension = p

  Number of total values stored in this vertex (number of state vector components).

Protected Attributes

- StateVector _states

  State vector with p states [p x 1].
- BackupStackType _backup

  backup stack (store and restore states).
- int _dimension = p

  Store dimension for all values stored (p states)
- FixedStates _fixed_states = FixedStates::Constant(false)

  Mask that stores fixed and unfixed states [p x 1] true: fixed, false: free.
- int _opt_vec_idx = -1

  Start-index in optimization vector (idx in hessian (row/column) and jacobian (column))

Friends

- bool operator== (const StateVertex< p > &lhs, const StateVertex< p > &rhs)
- bool operator!= (const StateVertex< p > &lhs, const StateVertex< p > &rhs)
- std::ostream & operator<< (std::ostream &os, const StateVertex< p > &state)

  Print state vector using std::ostream.

16.36.1 Detailed Description

template<int p>class teb::StateVertex< p >

This class wraps the discrete state vector $\mathbf{x}_k \in \mathbb{R}^p$ into a vertex type for the hyper-graph.

Author

Christoph Rösmann (`christoph.roesmann@tu-dortmund.de`)

See Also

ControlVertex, VertexType, TimeDiff, BaseEdge

Template Parameters

p	Number of states / Size of the state vector

Definition at line 25 of file teb_vertices.h.

16.36.2 Member Typedef Documentation

16.36.2.1 template<int p> using teb::StateVertex< p >::BackupStackType = std::stack<StateVector, std::vector<StateVector, Eigen::aligned_allocator<StateVector> > >

Definition at line 42 of file teb_vertices.h.

16.36.2.2 template<int p> using teb::StateVertex< p >::FixedStates = Eigen::Matrix<bool, p, 1>

Definition at line 36 of file teb_vertices.h.

16.36.2.3 template<int p> using teb::StateVertex< p >::ScalarType = Eigen::Matrix<double, 1, 1>

Definition at line 34 of file teb_vertices.h.

16.36.2.4 template<int p> using teb::StateVertex< p >::StateVector = Eigen::Matrix<double, p, 1>

Definition at line 32 of file teb_vertices.h.

16.36.2.5 using teb::VertexType::VertexContainer = std::vector<VertexType*> [inherited]

Definition at line 33 of file graph.h.

16.36.3 Constructor & Destructor Documentation

16.36.3.1 template<int p> teb::StateVertex< p >::StateVertex ( ) [inline]

Definition at line 50 of file teb_vertices.h.

16.36.3.2 template<int p> teb::StateVertex< p >::StateVertex (const Eigen::Ref< const StateVector > & *states*) [inline]

Parameters

states	Eigen::Vector with p states.

Definition at line 56 of file teb_vertices.h.

16.36.3.3 template<int p> teb::StateVertex< p >::StateVertex (const double ∗ *states*) [inline]

Parameters

states	double array with p states.

Definition at line 62 of file teb_vertices.h.

16.36.3.4 template<int p> teb::StateVertex< p >::StateVertex (const StateVertex< p > & *obj*) [inline]

Parameters

obj	Object of type StateVertex ∗

Definition at line 70 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states, and teb::StateVertex< p >::fixed_states().

16.36.4 Member Function Documentation

16.36.4.1 template<int p> virtual int teb::StateVertex< p >::dimension ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 79 of file teb_vertices.h.

16.36.4.2 template<int p> virtual int teb::StateVertex< p >::dimensionFree ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 81 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states.

16.36.4.3 template<int p> virtual void teb::StateVertex< p >::discardTop ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 213 of file teb_vertices.h.

References teb::StateVertex< p >::_backup.

16.36.4.4 template<int p> const FixedStates& teb::StateVertex< p >::fixed_states ( ) const [inline]

Returns

logical map (p x 1 vector of booleans

Definition at line 152 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states.

Referenced by teb::StateVertex< p >::operator=(), teb::StateVertex< p >::setFixedStates(), and teb::StateVertex< p >::StateVertex().

16.36.4.5 template<int p> virtual const double& teb::StateVertex< p >::getData (unsigned int *idx*) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 139 of file teb_vertices.h.

References teb::StateVertex< p >::_states.

16.36.4.6 template<int p> virtual void teb::StateVertex< p >::getDataFree (double * *target_vec*) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 126 of file teb_vertices.h.

References teb::StateVertex< p >::_states, and teb::StateVertex< p >::isFixedComp().

16.36.4.7 int teb::VertexType::getOptVecIdx ( ) const [inline], [inherited]

Returns

position of the first element of this vertex.

Definition at line 59 of file graph.h.

References teb::VertexType::_opt_vec_idx.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), and teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ().

16.36.4.8 template<int p> virtual bool teb::StateVertex< p >::isFixedAll ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 190 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states.

16.36.4.9 template<int p> bool teb::StateVertex< p >::isFixedAny ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 191 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states.

16.36.4.10 template<int p> virtual bool teb::StateVertex< p >::isFixedComp (int *idx*) const [inline], [virtual]

Parameters

idx	index to element in the StateVector

Returns

`true` if element with index `state_idx` is fixed.

Implements teb::VertexType.

Definition at line 198 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states.

Referenced by teb::StateVertex< p >::getDataFree(), teb::StateVertex< p >::plusFree(), and teb::StateVertex< p >::setFree().

16.36.4.11 template<int p> StateVertex<p>& teb::StateVertex< p >::operator= (const StateVertex< p > & *rhs*) [inline]

Definition at line 218 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states, teb::StateVertex< p >::_states, teb::StateVertex< p >::fixed_-states(), and teb::StateVertex< p >::states().

16.36.4.12 template<int p> virtual void teb::StateVertex< p >::plus (const VertexType ∗ *rhs*) [inline], [virtual]

Implements teb::VertexType.

Definition at line 84 of file teb_vertices.h.

References teb::StateVertex< p >::_states, and teb::StateVertex< p >::states().

16.36.4.13 template<int p> virtual void teb::StateVertex< p >::plus (const double ∗ *rhs*) [inline], [virtual]

Implements teb::VertexType.

Definition at line 91 of file teb_vertices.h.

References teb::StateVertex< p >::_states.

16.36.4.14 template<int p> virtual void teb::StateVertex< p >::plusFree (const double ∗ *rhs*) [inline], [virtual]

Implements teb::VertexType.

Definition at line 97 of file teb_vertices.h.

References teb::StateVertex< p >::_states, and teb::StateVertex< p >::isFixedComp().

16.36.4.15 template<int p> virtual void teb::StateVertex< p >::pop ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 203 of file teb_vertices.h.

References teb::StateVertex< p >::_backup, and teb::StateVertex< p >::top().

16.36.4.16 template<int p> virtual void teb::StateVertex< p >::push ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 202 of file teb_vertices.h.

References teb::StateVertex< p >::_backup, and teb::StateVertex< p >::_states.

16.36.4.17 template<int p> void teb::StateVertex< p >::setFixedAll (bool *fixed*) [inline]

Parameters

fixed	set to `true` in order to fix all *p* states

Definition at line 158 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states.

16.36.4.18 template<int p> void teb::StateVertex< p >::setFixedState (unsigned int *state_idx*, bool *fixed*) [inline]

Parameters

state_idx	index of the underlying StateVector component that should be fixed.
fixed	set to `true` in order to fix the selected state.

Definition at line 169 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states.

16.36.4.19 template<int p> void teb::StateVertex< p >::setFixedStates (bool *fixed*) [inline]

Parameters

fixed	set to `true` in order to fix all *p* states.

Definition at line 175 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states.

16.36.4.20 template<int p> void teb::StateVertex< p >::setFixedStates (const bool ∗ *fixed*) [inline]

Parameters

fixed	boolean array with *p* elements in which each `true` sets the corresponding state of the State-Vector to fixed.

Definition at line 181 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states.

16.36.4.21 template<int p> void teb::StateVertex< p >::setFixedStates (const Eigen::Ref< const FixedStates > & *fixed_states*) [inline]

Parameters

fixed_states	Eigen::Vector with *p* booleans in which each `true` sets the corresponding state of the State-Vector to fixed.

Definition at line 187 of file teb_vertices.h.

References teb::StateVertex< p >::_fixed_states, and teb::StateVertex< p >::fixed_states().

16.36.4.22 template<int p> virtual void **teb::StateVertex**< p >::setFree (const double * *rhs*) `[inline]`, `[virtual]`

Implements teb::VertexType.

Definition at line 111 of file teb_vertices.h.

References teb::StateVertex< p >::_states, and teb::StateVertex< p >::isFixedComp().

16.36.4.23 void teb::VertexType::setOptVecIdx (int *opt_vec_idx*) `[inline]`, `[inherited]`

Parameters

opt_vec_idx	index of the first value

Definition at line 53 of file graph.h.

References teb::VertexType::_opt_vec_idx.

16.36.4.24 template<int p> virtual void **teb::StateVertex**< p >::setStates (const Eigen::Ref< const **StateVector** > & *s*) `[inline]`, `[virtual]`

Parameters

s	state vector [p x 1]

Definition at line 150 of file teb_vertices.h.

References teb::StateVertex< p >::_states.

16.36.4.25 template<int p> virtual void **teb::StateVertex**< p >::setStates (double *state*) `[inline]`, `[virtual]`

Parameters

state	single state component.

Definition at line 151 of file teb_vertices.h.

References teb::StateVertex< p >::_states.

16.36.4.26 template<int p> virtual unsigned int **teb::StateVertex**< p >::stackSize ( ) const `[inline]`, `[virtual]`

Implements teb::VertexType.

Definition at line 214 of file teb_vertices.h.

References teb::StateVertex< p >::_backup.

16.36.4.27 template<int p> const **StateVector**& **teb::StateVertex**< p >::states ( ) const `[inline]`

Returns

state vector [p x 1]

Definition at line 148 of file teb_vertices.h.

References teb::StateVertex< p >::_states.

Referenced by teb::EdgeSystemDynamics< p, q, central >::computeValues(), teb::EdgeQuadraticForm< p, q >::computeValues(), teb::StateVertex< p >::operator=(), and teb::StateVertex< p >::plus().

16.36.4.28 template<int p> StateVector& teb::StateVertex< p >::states ( ) [inline]

Returns

state vector [p x 1]

Definition at line 149 of file teb_vertices.h.

References teb::StateVertex< p >::_states.

16.36.4.29 template<int p> virtual void teb::StateVertex< p >::top ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 208 of file teb_vertices.h.

References teb::StateVertex< p >::_backup, and teb::StateVertex< p >::_states.

Referenced by teb::StateVertex< p >::pop().

16.36.5 Friends And Related Function Documentation

16.36.5.1 template<int p> bool operator!= (const StateVertex< p > & *lhs*, const StateVertex< p > & *rhs*) [friend]

Definition at line 235 of file teb_vertices.h.

16.36.5.2 template<int p> std::ostream& operator<< (std::ostream & *os*, const StateVertex< p > & *state*) [friend]

Parameters

os	output stream object
state	vector specific StateVertex object

Returns

output stream object including a formatted string containing the state vector.

Definition at line 246 of file teb_vertices.h.

16.36.5.3 template<int p> bool operator== (const StateVertex< p > & *lhs*, const StateVertex< p > & *rhs*) [friend]

Definition at line 230 of file teb_vertices.h.

16.36.6 Member Data Documentation

16.36.6.1 template<int p> BackupStackType teb::StateVertex< p >::_backup [protected]

Definition at line 254 of file teb_vertices.h.

Referenced by teb::StateVertex< p >::discardTop(), teb::StateVertex< p >::pop(), teb::StateVertex< p >::push(), teb::StateVertex< p >::stackSize(), and teb::StateVertex< p >::top().

16.36.6.2 template<int p> int teb::StateVertex< p >::_dimension = p [protected]

Definition at line 256 of file teb_vertices.h.

16.36.6.3 template<int p> FixedStates teb::StateVertex< p >::_fixed_states = FixedStates::Constant(false) [protected]

Definition at line 257 of file teb_vertices.h.

Referenced by teb::StateVertex< p >::dimensionFree(), teb::StateVertex< p >::fixed_states(), teb::StateVertex< p >::isFixedAll(), teb::StateVertex< p >::isFixedAny(), teb::StateVertex< p >::isFixedComp(), teb::StateVertex< p >::operator=(), teb::StateVertex< p >::setFixedAll(), teb::StateVertex< p >::setFixedState(), teb::StateVertex< p >::setFixedStates(), and teb::StateVertex< p >::StateVertex().

16.36.6.4 int teb::VertexType::_opt_vec_idx = -1 [protected], [inherited]

Definition at line 69 of file graph.h.

Referenced by teb::VertexType::getOptVecIdx(), and teb::VertexType::setOptVecIdx().

16.36.6.5 template<int p> StateVector teb::StateVertex< p >::_states [protected]

Definition at line 253 of file teb_vertices.h.

Referenced by teb::StateVertex< p >::getData(), teb::StateVertex< p >::getDataFree(), teb::StateVertex< p >::operator=(), teb::StateVertex< p >::plus(), teb::StateVertex< p >::plusFree(), teb::StateVertex< p >::push(), teb::StateVertex< p >::setFree(), teb::StateVertex< p >::setStates(), teb::StateVertex< p >::states(), and teb::StateVertex< p >::top().

16.36.6.6 template<int p> const int teb::StateVertex< p >::Dimension = p [static]

Definition at line 29 of file teb_vertices.h.

16.36.6.7 template<int p> const int teb::StateVertex< p >::DimStates = p [static]

Definition at line 28 of file teb_vertices.h.

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

- teb_vertices.h

16.37 teb::SystemDynamics< p, q > Class Template Reference

Helper class for modeling nonlinear dynamic systems.

`#include <system_dynamics.h>`

Public Types

- using StateVector = Eigen::Matrix< double, p, 1 >

    Typedef for state vector with p states [p x 1].
- using ControlVector = Eigen::Matrix< double, q, 1 >

    Typedef for control input vector with q controls [q x 1].

Public Member Functions

- SystemDynamics ()

    Empty Constructor.
- virtual ∼SystemDynamics ()

    Empty Destructor.
- virtual StateVector stateSpaceModel (const Eigen::Ref< const StateVector > &x, const Eigen::Ref< const ControlVector > &u) const =0

    Implement the nonlinear state space model $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ here.

Static Public Attributes

- static const int DimStates = p

    Store number of states as static variable.
- static const int DimControls = q

    Store number of control inputs as static variable.

Protected Member Functions

- StateVector finiteElemFwdEuler (const Eigen::Ref< const StateVector > &x1, const Eigen::Ref< const StateVector > &x2, double dt)

    xdot = stateSpaceModel(x,u)
- StateVector finiteElemCentralDiff (const Eigen::Ref< const StateVector > &x_km1, const Eigen::Ref< const StateVector > &x_kp1, double dt)

    Central difference approximation for continuous-time derivatives.

Friends

- template<int pf, int qf, bool cf>
  class EdgeSystemDynamics
- template<int pf, int qf>
  class Simulator

16.37.1 Detailed Description

template<int p, int q>class teb::SystemDynamics< p, q >

This abstract class provides an interface for modeling dynamic systems utilizing nonlinear state space equations:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

$\mathbf{x} \in \mathbb{R}^p$ denotes the state vector and $\mathbf{u} \in \mathbb{R}^q$ denotes the control input vector. The system dynamics are modeled in continuous time-domain and discretized by the program itself.

- E.g.the TebController class discretizes the system equations using teb::FiniteDifferences (see EdgeSystem-Dynamics, finiteElemFwdEuler() and finiteElemCentralDiff()).
- And the Simulator class is able to apply different numerical integration schemes to the state space model implemented here (see NumericalIntegrator).

Todo Methods/Intefrace for jacobian and hessian calculation

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

Template Parameters

p	Number of state variables
q	Number of input variables

Examples:

integrator_system1.cpp, integrator_system2.cpp, integrator_system_classic_mpc.cpp, integrator_system_mex.cpp, integrator_system_sfun.cpp, linear_system_ode.cpp, linear_system_ode_ctrl_comparison.cpp, linear_system_state_space.cpp, mobile_robot_teb.cpp, rocket_system.cpp, and van_der_pol_system.cpp.

Definition at line 34 of file system_dynamics.h.

16.37.2 Member Typedef Documentation

16.37.2.1 template<int p, int q> using teb::SystemDynamics< p, q >::ControlVector = Eigen::Matrix<double, q, 1>

Definition at line 50 of file system_dynamics.h.

16.37.2.2 template<int p, int q> using teb::SystemDynamics< p, q >::StateVector = Eigen::Matrix<double, p, 1>

Definition at line 48 of file system_dynamics.h.

16.37.3 Constructor & Destructor Documentation

16.37.3.1 template<int p, int q> teb::SystemDynamics< p, q >::SystemDynamics ( ) [inline]

Definition at line 52 of file system_dynamics.h.

16.37.3.2 template<int p, int q> virtual teb::SystemDynamics< p, q >::~SystemDynamics ( ) [inline], [virtual]

Definition at line 53 of file system_dynamics.h.

16.37.4 Member Function Documentation

16.37.4.1 template<int p, int q> StateVector teb::SystemDynamics< p, q >::finiteElemCentralDiff (const Eigen::Ref< const StateVector > & *x_km1*, const Eigen::Ref< const StateVector > & *x_kp1*, double *dt*) [inline], [protected]

Approximate derivatives using $\dot{x}(t) = \frac{x_{k+1} - x_{k-1}}{2\Delta T}$

Parameters

x_km1	Discrete state vector $\mathbf{x}_{k-1}$ [p x 1]
x_kp1	Discrete state vector $\mathbf{x}_{k+1}$ [p x 1]
dt	Time interval / discretization step width ΔT

Returns

Finite difference approximation

Definition at line 107 of file system_dynamics.h.

16.37.4.2 template< int p, int q > StateVector teb::SystemDynamics< p, q >::finiteElemFwdEuler (const Eigen::Ref< const StateVector > & *x1*, const Eigen::Ref< const StateVector > & *x2*, double *dt*) [inline], [protected]

Forward difference approximation for continuous-time derivatives

Approximate derivatives using $\dot{x}(t) = \frac{x_{k+1} - x_k}{\Delta T}$

Parameters

x1	Discrete state vector $\mathbf{x}_k$ [p x 1]
x2	Discrete state vector $\mathbf{x}_{k+1}$ [p x 1]
dt	Time interval / discretization step width ΔT

Returns

Finite difference approximation

Definition at line 91 of file system_dynamics.h.

16.37.4.3 template< int p, int q > virtual StateVector teb::SystemDynamics< p, q >::stateSpaceModel (const Eigen::Ref< const StateVector > & *x*, const Eigen::Ref< const ControlVector > & *u*) const [pure virtual]

This method defines the state space model of the dedicated dynamic system.

Implement only the right hand side $f(\cdot)$ of $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ and return the equation as a StateVector type.

E.g. $\dot{x} = x + u \quad x, u \in \mathbb{R}$ (with `p` = 1 and `q` = 1)

```
StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
{
     StateVector f = x[0] + u[0];
     return f;
}
```

Parameters

x	State vector $\mathbf{x}$ [p x 1]
u	Control input vector $\mathbf{u}$ [q x 1]

Returns

$f(\mathbf{x}, \mathbf{u})$ as vector [p x 1]

Examples:

integrator_system1.cpp, integrator_system2.cpp, integrator_system_classic_mpc.cpp, integrator_system_mex.cpp, integrator_system_sfun.cpp, linear_system_ode.cpp, linear_system_ode_ctrl_comparison.cpp, linear_system_state_space.cpp, mobile_robot_teb.cpp, rocket_system.cpp, and van_der_pol_system.cpp.

Referenced by teb::ExplicitEuler< p, q >::integrate(), teb::RungeKuttaClassic< p, q >::integrate(), and teb::RungeKutta5thOrder< p, q >::integrate().

16.37.5 Friends And Related Function Documentation

16.37.5.1 template<int p, int q> template<int pf, int qf, bool cf> friend class EdgeSystemDynamics [friend]

Definition at line 37 of file system_dynamics.h.

16.37.5.2 template<int p, int q> template<int pf, int qf> friend class Simulator [friend]

Definition at line 40 of file system_dynamics.h.

16.37.6 Member Data Documentation

16.37.6.1 template<int p, int q> const int teb::SystemDynamics< p, q >::DimControls = q [static]

Definition at line 45 of file system_dynamics.h.

16.37.6.2 template<int p, int q> const int teb::SystemDynamics< p, q >::DimStates = p [static]

Definition at line 44 of file system_dynamics.h.

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

- system_dynamics.h

16.38 teb::Config::Teb Struct Reference

Configurations related to the TEB algorithm.

```
#include <config.h>
```

Public Attributes

- unsigned int n_min = 3

  Minimum number of samples (states and control input pairs) allowed.
- unsigned int n_pre = 20

  Initial number of samples (used in TebController::updateGoal()).
- unsigned int n_max = 150

  Maximum number of samples allowed.
- double dt_min = 0.01

  Minimum dt (only considered if EdgePositiveTime is activated).
- double dt_ref = 0.1

  Desired reference sampling time (the TEB tries to converge towards dt_ref) (make sure dt_ref>0).
- double dt_hyst = 0.01

  Sampling time hysteresis used in resizeTEB to avoid oscillations (default: 0.1∗dt_ref).
- double goal_dist_force_reinit = 0.5

  Force reinit of the trajectory if distance between current and new goal is above threshold.
- unsigned int teb_iter = 5

  Number of outer-loop TEB iterations (used for guiding ΔT towards dt_ref, see TebController::optimizeTEB()).

- FiniteDifferences diff_method = FiniteDifferences::FORWARD

  Select finite difference method for discretizing system dynamics and obtaining ΔT.

16.38.1 Detailed Description

Definition at line 60 of file config.h.

16.38.2 Member Data Documentation

16.38.2.1 FiniteDifferences teb::Config::Teb::diff_method = FiniteDifferences::FORWARD

Definition at line 70 of file config.h.

16.38.2.2 double teb::Config::Teb::dt_hyst = 0.01

Definition at line 67 of file config.h.

16.38.2.3 double teb::Config::Teb::dt_min = 0.01

Examples:

mobile_robot_teb.cpp.

Definition at line 65 of file config.h.

Referenced by teb::TebController$<$ p, q $>$::setupHorizon().

16.38.2.4 double teb::Config::Teb::dt_ref = 0.1

Definition at line 66 of file config.h.

Referenced by teb::TebController$<$ p, q $>$::resetController(), and teb::TebController$<$ p, q $>$::setupHorizon().

16.38.2.5 double teb::Config::Teb::goal_dist_force_reinit = 0.5

Definition at line 68 of file config.h.

16.38.2.6 unsigned int teb::Config::Teb::n_max = 150

Examples:

mobile_robot_teb.cpp.

Definition at line 64 of file config.h.

16.38.2.7 unsigned int teb::Config::Teb::n_min = 3

Examples:

mobile_robot_teb.cpp.

Definition at line 62 of file config.h.

Referenced by teb::TebController$<$ p, q $>$::setupHorizon().

16.38.2.8 unsigned int teb::Config::Teb::n_pre = 20

Examples:

mobile_robot_teb.cpp.

Definition at line 63 of file config.h.

Referenced by teb::TebController< p, q >::setupHorizon().

16.38.2.9 unsigned int teb::Config::Teb::teb_iter = 5

Definition at line 69 of file config.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights().

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

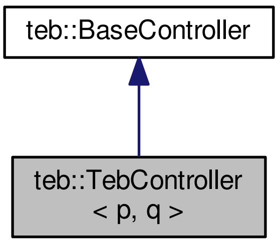
- config.h

16.39 teb::TebController< p, q > Class Template Reference

Main Timed-Elastic-Band controller class.

`#include <teb_controller.h>`

Inheritance diagram for teb::TebController< p, q >:



Public Types

- using StateVector = Eigen::Matrix< double, p, 1 >

    Typedef for state vector with p states [p x 1].
- using ControlVector = Eigen::Matrix< double, q, 1 >

    Typedef for control input vector with q controls [q x 1].
- using StateSequence = std::deque< StateVertex< p > >

    Typedef for the state sequence that contains all state vertices.
- using ControlSequence = std::deque< ControlVertex< q > >

    Typedef for the control sequence that contains all control vertices.
- using EdgeContainer = typename EdgeType::EdgeContainer

    Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).

- using VertexContainer = typename VertexType::VertexContainer

    Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).
- using FixedStates = Eigen::Matrix< bool, p, 1 >

    Typedef to represent a fixed/unfixed boolean vector w.r.t. states (StateVector) [p x 1].

Public Member Functions

- TebController (BaseSolver *solver=nullptr)

    Default constructor.
- TebController (const Config *config, BaseSolver *solver=nullptr)

    Extended constructor with config pointer.
- virtual ∼TebController ()

    !
- virtual Eigen::VectorXd firstState () const

    Access first state of the TEB sequence.
- const StateVector & firstStateRef () const

    Access first state of the TEB sequence.
- virtual Eigen::VectorXd lastState () const

    Access last state of the TEB sequence.
- const StateVector & lastStateRef () const

    Access last state of the TEB sequence.
- virtual Eigen::VectorXd firstControl () const

    Access first control input of the TEB sequence (used for MPC)
- const ControlVector & firstControlRef () const

    Access first control input of the TEB sequence (used for MPC)
- Eigen::VectorXd firstControlSaturated () const

    Access saturated first control inputs of the TEB sequence (used for MPC)
- const StateSequence & stateSequence () const

    Access complete state sequence container.
- const ControlSequence & controlSequence () const

    Access complete control input sequence container.
- const TimeDiff & dt () const

    Access vertex that stores the TEB time difference ΔT.
- virtual double getDt () const

    Return the TEB time difference ΔT.
- virtual unsigned int getN () const

    Get current number of discrete state vectors in the state sequence.
- virtual unsigned int getM () const

    Get current number of discrete control vectors in the control input sequence.
- virtual Eigen::MatrixXd returnControlInputSequence () const

    Return the complete control input sequence $u_k, k = 1, \ldots, n-1$.
- const HyperGraph & graph () const

    Access optimizable hyper-graph instance (read-only).
- virtual void initTrajectory (const Eigen::Ref< const StateVector > &x0, const Eigen::Ref< const StateVector > &xf, unsigned int n)

    Initialize trajectory between start $x0$ and goal xf with n discrete samples.
- void resizeTrajectory ()

    Resize existing trajectory according to current sample time ΔT.
- void resampleTrajectory (unsigned int n_new)

    Resample trajectory using linear interpolation.

- virtual void updateStart (const Eigen::Ref< const StateVector > &x0)

  Set new start state.
- virtual void updateGoal (const Eigen::Ref< const StateVector > &xf)

  Set new final state.
- void removeStateControlInputPair (unsigned int sample_no, bool predict_control=true)

  Remove StateVector and ControlVector at index `sample_no` from the state and control sequences.
- void insertStateControlInputPair (unsigned int sample_idx, const Eigen::Ref< const StateVector > &state, const Eigen::Ref< const ControlVector > &control, bool predict_control=true)

  Insert StateVector and ControlVector to the state and contorl sequences at index `sample_idx`.
- void predictControl (Eigen::Ref< ControlVector > ctrl_out, const Eigen::Ref< const StateVector > &x1, const Eigen::Ref< const StateVector > &x2)

  Predict control `u` that corresponds to the transition from `x1` to `x2`.
- void pushBackStateControlInputPair (const Eigen::Ref< const StateVector > &state, const Eigen::Ref< const ControlVector > &control)

  Add a state vector $\mathbf{x}_k$ and a control input $\mathbf{u}_k$ to the end of the state and control sequences.
- void pushFrontStateControlInputPair (const Eigen::Ref< const StateVector > &state, const Eigen::Ref< const ControlVector > &control)

  Add a state vector $\mathbf{x}_k$ and a control input $\mathbf{u}_k$ to the beginning of the state and control sequences.
- void setGoalStatesFixedOrUnfixed (const bool *fixed_goal_states)

  Set individual goal state vector components fixed or unfixed for optimization.
- void setGoalStatesFixedOrUnfixed (const Eigen::Ref< const Eigen::Matrix< bool, p, 1 >> &fixed_goal_-states)

  Set individual goal state vector components fixed or unfixed for optimization (Eigen version)
- void setGoalStatesFixedOrUnfixed (bool all_goal_states_fixed)

  Set all goal state vector components fixed or unfixed for optimization.
- void setupHorizon (bool fixed_goal, bool fixed_resolution, int no_samples=-1, double dt=-1)

  Setup horizon for the underlying optimal control problem.
- virtual void resetController ()

  Delete all Vertices (State and control input sequences), all Edges (Hyper-Graph, clearEdges()) and reset ΔT to Config::Teb::dt_ref.
- virtual void buildOptimizationGraph ()

  Build Hyper-Graph for graph based optimization.
- virtual void customOptimizationGraph ()

  Override to add or modify the hyper-graph created in buildOptimizationGraph()
- virtual void customOptimizationGraphHotStart ()

  Use this method to apply some changes before hot-starting from previous graph structures.
- void initOptimization ()

  Initialize optimiziation - run before calling optimzieTEB().
- void optimizeTEB ()

  TEB optimization function (includes outer loop and BaseSolver calls).
- virtual void step (const double *const x0, const double *const xf, double *ctrl_out=nullptr)

  Perform complete MPC step (initialization if necessary or update and optimization)
- void step (const Eigen::Ref< const StateVector > &x0, const Eigen::Ref< const StateVector > &xf, Eigen::-Map< ControlVector > *ctrl_out=nullptr)

  Perform complete MPC step (initialization if necessary or update and optimization) (Eigen version)
- void activateObjectiveTimeOptimal (bool activate=true)

  Activate predefined edge for time-optimal control.
- void activateObjectiveQuadraticForm (double Q, double R, double Qf, bool activate=true)

  Activate predefined edge for minimizing the quadratic form.
- void activateControlBounds (const Eigen::Ref< const ControlVector > u_min, const Eigen::Ref< const ControlVector > u_max, bool activate=true)

  Activate predefined edge for control input bounds.

- void activateControlBounds (unsigned int control_idx, double u_min, double u_max, bool activate=true)

  Activate predefined edge for control input bounds (single component version)
- void activateStateBounds (const Eigen::Ref$<$ const StateVector $>$ &x_min, const Eigen::Ref$<$ const StateVector $>$ &x_max, bool activate=true)

  Activate predefined edge for state bounds.
- void activateStateBounds (unsigned int state_idx, double x_min, double x_max, bool activate=true)

  Activate predefined edge for state bounds (single component version)
- void setFixedDt (bool fixed)

  Fix the TimeDiff ΔT during optimization.
- void setSystemDynamics (SystemDynamics$<$ p, q $>$ $*$system, bool activate=true)

  Set system dynamics and activate the predifined equality-constraint-edge.
- void setSolver (BaseSolver $*$solver)

  Set algorithm to solve the underlying optimization problem.
- Eigen::VectorXd getAbsoluteTimeVec () const

  Get vector of absolute times $t = [0, \Delta T, 2\Delta T, \ldots, n\Delta T]$.
- Eigen::MatrixXd getStateCtrlInfoMat () const

  Get TEB states and control inputs in matrix form (Use for debugging).
- Eigen::Matrix$<$ bool, p+q,-1 $>$ getTEBFixedMap () const

  Get TEB matrix as bool matrix: `true:` fixed, `false` = free (Use for debugging).
- Eigen::Matrix$<$ double,-1, 2 $>$ getSampleOptVecIdxVMat () const

  Get Hessian index (col 1) and number of free variables (col 2) for each VertexType (StateVertex, ControlVertex, TimeDiff) (rowwise).

Public Attributes

- const Config $*$ cfg

  Store pointer to the Config class.

Static Public Attributes

- static const unsigned int NoStates = p

  Store number of states as static variable.
- static const unsigned int NoControls = q

  Store number of control inputs as static variable.

Protected Member Functions

- void setFixedStart (bool fixed)

  Fix the complete first StateVector of the state sequence (start state vector).
- void setFixedStart (const bool $*$fixed)

  Fix selected components of the first StateVector of the state sequence (start state vector).
- void setFixedGoal (bool fixed)

  Fix the complete last StateVector of the state sequence (final/goal state vector).
- void setFixedGoal (const bool $*$fixed)

  Fix selected components of the last StateVector of the state sequence (final state vector).
- void setFixedGoal (const Eigen::Ref$<$ const Eigen::Matrix$<$ bool, p, 1 $>>$ &fixed)

  Fix selected components of the last StateVector of the state sequence (final state vector).
- void getActiveVertices ()

  Collect active vertices and determine the Hessian start index for each vertex (StateVertex, ControlVertex and TimeDiff).

Protected Attributes

- StateSequence _state_seq

  Store the complete sequence of state vectors (represented as StateVertex for the representation in a hyper-graph)
- ControlSequence _ctrl_seq

  Store the complete sequence of contorl input vectors (represented as ControlVertex for the representation in a hyper-graph)
- TimeDiff _dt

  TimeDiff vertex that represents the required transition time between consecutive states and control inputs.
- StateVector _goal_backup

  Store and backup goal vector (should be changed in all functions that set the goal point).
- int _no_samples = -1

  Number of samples for trajectory initialization (if -1, Config::Teb::n_pre is used, see setupHorizon()).
- double _dt_ref = -1

  Reference time difference (discretization) (if -1, Config::Teb::dt_ref is used, see setupHorizon()).
- bool _optimized

  Status flag that is `true`*, if at least one optimization step in optimizeTEB() is performed before.*
- HyperGraph _graph

  Instance of the optimization hyper-graph to store all active vertices and edges (interface for the solver)
- FixedStates _fixed_goal_states = FixedStates::Constant(true)

  Store information about (un-)/fixed components of the goal StateVector (initialized to fixed, see buildOptimization-Graph()).
- BaseSolver ∗ _solver = nullptr

  Pointer to solver class specified using setSolver() or TebController().
- bool _cfg_owned = false

  If `true`*, the Config TebController::cfg is owned by this class, that requires its deletion inside ∼TebController().*
- bool _active_time_optimal = false

  if `true`*, EdgeMinimizeTime is activated (see activateObjectiveTimeOptimal()).*
- std::tuple< bool, double,
  double, double > _active_quadratic_form = std::make_tuple(false,1,1,1)

  Stores activation status of EdgeQuadraticForm.
- std::pair< bool,
  SystemDynamics< p, q > ∗ > _active_system_dynamics = std::make_pair(false,nullptr)

  Stores activation status of EdgeSystemDynamics and a pointer to the problem specific SystemDynamics object.
- std::tuple< bool,
  ControlVector, ControlVector > _active_control_bounds = std::make_tuple(false,ControlVector::Constant(IN-F),ControlVector::Constant(-INF))

  Stores activation status of EdgeControlBounds and mininum/maximum bounds on each ControlVector.
- std::tuple< bool, StateVector,
  StateVector > _active_state_bounds = std::make_tuple(false, StateVector::Constant(-INF), StateVector::-Constant(INF))

  Stores activation status of EdgeStateBounds and mininum/maximum bounds on each StateVector.

16.39.1 Detailed Description

template<int p, int q>class teb::TebController< p, q >

This class defines the main TEB controller functions required to controll a dynamic system.

The underlying optimziation problem is defined as a hyper-graph in which edges define cost/objective functions (see TebController::buildOptimizationGraph and TebController::customOptimizationGraph class methods).

To overcome the creation of custom hyper-graphs, one can use the SystemDynamics class and pass the system to this class using TebController::setSystemDynamics. Other predifined edges are listed below.

The TEB controller requires a dedicated solver (inherited from BaseSolver) that can be passed using the setSolver method.

Author

Christoph Rösmann (`christoph.roesmann@tu-dortmund.de`)

Todo Using the TebController inside a standard container (e.g. vector) does not work as supposed to do.

Create copy and move constructors.

See Also

TebController::buildOptimizationGraph, TebController::customOptimizationGraph, SystemDynamics, TebController::setSystemDynamics, TebController::activateObjectiveTimeOptimal, TebController::activateControlBounds, TebController::activateStateBounds, TebController::setSolver BaseSolver

Template Parameters

p	Number of states / Size of the state vector
q	Number of control inputs / Size of the control vectorp: number of states, q: number of control inputs

Examples:

integrator_system1.cpp, integrator_system2.cpp, integrator_system_classic_mpc.cpp, integrator_system_mex.cpp, integrator_system_sfun.cpp, linear_system_ode.cpp, linear_system_ode_ctrl_comparison.cpp, linear_system_state_space.cpp, rocket_system.cpp, and van_der_pol_system.cpp.

Definition at line 54 of file teb_controller.h.

16.39.2 Member Typedef Documentation

16.39.2.1 template< int p, int q > using **teb::TebController**< p, q >**::ControlSequence** = std::deque< **ControlVertex**<q> >

Definition at line 71 of file teb_controller.h.

16.39.2.2 template< int p, int q > using **teb::TebController**< p, q >**::ControlVector** = Eigen::Matrix<double, q, 1>

Definition at line 67 of file teb_controller.h.

16.39.2.3 template< int p, int q > using **teb::TebController**< p, q >**::EdgeContainer** = typename **EdgeType::EdgeContainer**

Definition at line 73 of file teb_controller.h.

16.39.2.4 template< int p, int q > using **teb::TebController**< p, q >**::FixedStates** = Eigen::Matrix<bool,p,1>

Definition at line 77 of file teb_controller.h.

16.39.2.5 template< int p, int q > using **teb::TebController**< p, q >**::StateSequence** = std::deque< **StateVertex**<p> >

Definition at line 69 of file teb_controller.h.

16.39.2.6 template<int p, int q> using **teb::TebController**< p, q >**::StateVector** = Eigen::Matrix<double,p,1>

Definition at line 65 of file teb_controller.h.

16.39.2.7 template<int p, int q> using **teb::TebController**< p, q >**::VertexContainer** = typename **VertexType::VertexContainer**

Definition at line 75 of file teb_controller.h.

16.39.3 Constructor & Destructor Documentation

16.39.3.1 template<int p, int q> **teb::TebController**< p, q >**::TebController** (**BaseSolver** * *solver* = `nullptr`) `[inline]`

This constructor creates a default Config object for TEB and solver settings.

Parameters

solver	Register solver type as subclass of BaseSolver. If not passed, you must use setSolver() later.

Definition at line 101 of file teb_controller.h.

References teb::TebController< p, q >::_cfg_owned.

16.39.3.2 template<int p, int q> **teb::TebController**< p, q >**::TebController** (const **Config** * *config*, **BaseSolver** * *solver* = `nullptr`) `[inline]`

Pass config class to controller and solver. TebController does not free Config afterwards.

Parameters

config	Set user-defined config for the controller and solver.
solver	Register solver type as subclass of BaseSolver. If not passed, you must use setSolver() later.

Definition at line 111 of file teb_controller.h.

References teb::TebController< p, q >::setSolver().

16.39.3.3 template<int p, int q> virtual **teb::TebController**< p, q >**::~TebController** ( ) `[inline]`, `[virtual]`

Destructor. Memory is freed if necessary.

Definition at line 119 of file teb_controller.h.

References teb::TebController< p, q >::_cfg_owned, teb::TebController< p, q >::_graph, teb::TebController< p, q >::cfg, and teb::HyperGraph::clearGraph().

16.39.4 Member Function Documentation

16.39.4.1 template<int p, int q> void **teb::TebController**< p, q >**::activateControlBounds** (const Eigen::Ref< const **ControlVector** > *u_min*, const Eigen::Ref< const **ControlVector** > *u_max*, bool *activate* = `true`) `[inline]`

Adds the following inequality constraints to the optimization problem:

- $\mathbf{u}_k \leq u_{max}$
- $\mathbf{u}_k \geq u_{min}$

Parameters

u_min	minimum bound on the control input $\mathbf{u}_k$
u_max	maximum bound on the control input $\mathbf{u}_k$
activate	if set to `true` than the edge is activated.

See Also

buildOptimizationGraph(), activateObjectiveTimeOptimal(), activateStateBounds(), setSystemDynamics()

Definition at line 481 of file teb_controller.h.

References teb::TebController< p, q >::_active_control_bounds, teb::TebController< p, q >::_graph, and teb::HyperGraph::notifyGraphModified().

16.39.4.2 template<int p, int q> void **teb::TebController**< p, q >::activateControlBounds (unsigned int *control_idx*, double *u_min*, double *u_max*, bool *activate* = `true`) `[inline]`

Adds the following inequality constraints to the optimization problem:

- $\mathbf{u}_k \leq \mathbf{u}_{max}$ (component-wise)
- $\mathbf{u}_k \geq \mathbf{u}_{min}$ (component-wise)

The default initialization of the bounds is `-INF` and `INF` respectively, if this predefined edge is activated. This version of activateControlBounds() makes use of the default initialization and just updates the bounds for index `state_index`. All other bounds remain untouched (and if not changed before, they are bound to `-INF` and `INF`.

Parameters

control_idx	selected component of $\mathbf{x}_k$ that should be bounded to `x_min` and `x_max`.
u_min	minimum bound on the component with index `control_idx` of control input $\mathbf{u}_k$.
u_max	maximum bound on the component with index `control_idx` of the control input $\mathbf{u}_k$.
activate	if set to `true` than the edge is activated.

Todo Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

See Also

buildOptimizationGraph(), activateObjectiveTimeOptimal(), activateControlBounds(), setSystemDynamics()

Definition at line 510 of file teb_controller.h.

References teb::TebController< p, q >::_active_control_bounds, teb::TebController< p, q >::_graph, and teb::HyperGraph::notifyGraphModified().

16.39.4.3 template<int p, int q> void **teb::TebController**< p, q >::activateObjectiveQuadraticForm (double *Q*, double *R*, double *Qf*, bool *activate* = `true`) `[inline]`

Adds the following cost function

$$J = (\mathbf{x}_n - \mathbf{x}_f)^T \mathbf{Q}_f (\mathbf{x}_n - \mathbf{x}_f + \sum_{k=0}^{n-1} ((\mathbf{x}_k - \mathbf{x}_f)^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_f) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k)$$

to the global cost/objective function.

The weight matrices are assumed to be diagonal matrices with unifrom values Q, R and Qf.

See EdgeQuadraticForm for further details.

Parameters

Q	Uniform scalar weight for minimizing the squared state error
R	Uniform scalar weight for minimizing the squared control input
Qf	Uniform scalar weight for minimizing the squared state error for the final state (last state of the horizon)
activate	if set to `true` than the edge is activated.

See Also

buildOptimizationGraph(), activateObjectiveTimeOptimal(), activateControlBounds(), activateStateBounds(), setSystemDynamics()

Definition at line 460 of file teb_controller.h.

References teb::TebController< p, q >::_active_quadratic_form, teb::TebController< p, q >::_graph, and teb::-HyperGraph::notifyGraphModified().

16.39.4.4 template<int p, int q> void **teb::TebController**< p, q >::activateObjectiveTimeOptimal (bool *activate* = `true`) `[inline]`

Adds the following cost function $V(\mathscr{B}) = (n-1)\Delta T$ to the global cost/objective function.

Additionally, the inequality constraint $\Delta T > \Delta T_{min}$ is added. ΔT_{min} can be changed in Config::Teb::teb_min.

Parameters

activate	if set to `true` than the edge is activated.

See Also

buildOptimizationGraph(), activateObjectiveQuadraticForm(), activateControlBounds(), activateState-Bounds(), setSystemDynamics()

Definition at line 435 of file teb_controller.h.

References teb::TebController< p, q >::_active_time_optimal, teb::TebController< p, q >::_graph, and teb::Hyper-Graph::notifyGraphModified().

16.39.4.5 template<int p, int q> void **teb::TebController**< p, q >::activateStateBounds (const Eigen::Ref< const **StateVector** > & *x_min*, const Eigen::Ref< const **StateVector** > & *x_max*, bool *activate* = `true`) `[inline]`

Adds the following inequality constraints to the optimization problem:

- $\mathbf{x}_k \leq \mathbf{x}_{max}$ (component-wise)
- $\mathbf{x}_k \geq \mathbf{x}_{min}$ (component-wise)

To just bound a subset of components of the state $\mathbf{x}_k$, set unbounded components to -$\mathrm{Inf} \leq x_k \leq \mathrm{Inf}$ using the `INF` macro. Or see overloaded function activateStateBounds().

Parameters

x_min	minimum bounds on the state $\mathbf{x}_k$ [p x 1].
x_max	maximum bounds on the state $\mathbf{x}_k$ [p x 1].

activate	if set to `true` than the edge is activated.

Todo Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

See Also

buildOptimizationGraph(), activateObjectiveTimeOptimal(), activateControlBounds(), setSystemDynamics()

Definition at line 539 of file teb_controller.h.

References teb::TebController< p, q >::_active_state_bounds, teb::TebController< p, q >::_graph, and teb::HyperGraph::notifyGraphModified().

16.39.4.6 template<int p, int q> void teb::TebController< p, q >::activateStateBounds (unsigned int *state_idx*, double *x_min*, double *x_max*, bool *activate* = `true`) `[inline]`

Adds the following inequality constraints to the optimization problem:

- $\mathbf{x}_k \leq \mathbf{x}_{max}$ (component-wise)
- $\mathbf{x}_k \geq \mathbf{x}_{min}$ (component-wise)

The default initialization of the bounds is `-INF` and `INF` respectively, if this predefined edge is activated. This version of activateStateBounds() makes use of the default initialization and just updates the bounds for index `state_index`. All other bounds remain untouched (and if not changed before, they are bound to `-INF` and `INF`.

Parameters

state_idx	selected component of $\mathbf{x}_k$ that should be bounded to `x_min` and `x_max`.
x_min	minimum bound on the component with index `state_idx` of state $\mathbf{x}_k$.
x_max	maximum bound on the component with index `state_idx` of the state $\mathbf{x}_k$.
activate	if set to `true` than the edge is activated.

Todo Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

See Also

buildOptimizationGraph(), activateObjectiveTimeOptimal(), activateControlBounds(), setSystemDynamics()

Definition at line 568 of file teb_controller.h.

References teb::TebController< p, q >::_active_state_bounds, teb::TebController< p, q >::_graph, and teb::HyperGraph::notifyGraphModified().

16.39.4.7 template<int p, int q> void teb::TebController< p, q >::buildOptimizationGraph ( ) `[virtual]`

Construct the hyper-graph for the underlying optimization problem.

Instantiate edges as subclasses of BaseEdge and add them to the corresponding containers.

- Add objective/cost function edges to TebController::_graph via HyperGraph::addEdgeObjective().
- equality constraints to TebController::_graph via HyperGraph::addEdgeEquality().
- and inequality constraints to TebController::_graph via HyperGraph::addEdgeInequality().

Example for the objective EdgeMinimizeTime that is derived from BaseEdge:

```
EdgeMinimizeTime* to_edge = new EdgeMinimizeTime; // Instantiate new edge object.
 to_edge->setVertex(&_dt); // Connect edge with vertex TimeDiff.
 to_edge->setData(getN()-1); // Set weight/data (only for EdgeMinimizeTime).
 _graph.addEdgeObjective(to_edge); // Store pointer to the container.
```

The implementation contains a few default edges commonly used in MPC and Optimal Control scenarios. See:

- activateObjectiveTimeOptimal()
- activateObjectiveQuadraticForm()
- activateControlBounds()
- activateStateBounds()
- setSystemDynamics()

After checking and adding activated default edges, the function calls customOptimizationGraph() for extended user supplied edges.

This method is called within initOptimization().

See Also

customOptimizationGraph, customOptimizationGraphHotStart, HyperGraph, BaseEdge

Todo Here we need to fix the last control input since it undefined (no control is needed to go to state n+1). Maybe we can change the code somewhere else, that the contorl sequence is always getN()-1.

Definition at line 590 of file teb_controller.hpp.

References teb::FORWARD, teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds(), teb-::EdgeMinimizeTime::setData(), teb::EdgeQuadraticForm< p, q >::setReference(), and teb::EdgeQuadraticForm< p, q >::setWeights().

16.39.4.8 template<int p, int q> const ControlSequence& teb::TebController< p, q >::controlSequence ( ) const [inline]

Returns

Reference to the control input sequence (read-only)

Definition at line 186 of file teb_controller.h.

References teb::TebController< p, q >::_ctrl_seq.

16.39.4.9 template<int p, int q> virtual void teb::TebController< p, q >::customOptimizationGraph ( ) [inline], [virtual]

Derive TEB class and override customOptimizationGraph() to manually construct the hyper-graph or add edges in addition to activated common edges in buildOptimizationGraph(). See the description of buildOptimizationGraph() for more information on how to add edges.

See Also

buildOptimizationGraph, customOptimizationGraphHotStart, HyperGraph, BaseEdge, initOptimization()

Definition at line 375 of file teb_controller.h.

16.39.4.10 template< int p, int q > virtual void teb::TebController< p, q >::customOptimizationGraphHotStart ( ) [inline], [virtual]

Depending on HyperGraph::isGraphModified() the graph is either cleared and constructed again or the previous graph is used. In the latter case customOptimizationGraph(), buildOptimizationGraph() and getActiveVertices() are not called. If you wish to apply changes without recreating the strucutre add them to this function. It is called inside initOptimization().

See Also

customOptimizationGraph, buildOptimizationGraph, HyperGraph, BaseEdge, initOptimization()

Definition at line 386 of file teb_controller.h.

16.39.4.11 template< int p, int q > const TimeDiff& teb::TebController< p, q >::dt ( ) const [inline]

Returns

Reference to time-diff vertex (read-only)

Definition at line 191 of file teb_controller.h.

References teb::TebController< p, q >::_dt.

Referenced by teb::TebController< p, q >::setupHorizon().

16.39.4.12 template< int p, int q > virtual Eigen::VectorXd teb::TebController< p, q >::firstControl ( ) const [inline], [virtual]

Returns

Copy of the first control input (read-only)

Implements teb::BaseController.

Definition at line 150 of file teb_controller.h.

References teb::TebController< p, q >::_ctrl_seq.

Referenced by teb::TebController< p, q >::firstControlSaturated(), and teb::TebController< p, q >::step().

16.39.4.13 template< int p, int q > const ControlVector& teb::TebController< p, q >::firstControlRef ( ) const [inline]

Returns

Reference to the first control input (read-only)

Definition at line 155 of file teb_controller.h.

References teb::TebController< p, q >::_ctrl_seq.

16.39.4.14 template< int p, int q > Eigen::VectorXd teb::TebController< p, q >::firstControlSaturated ( ) const [inline]

This function saturates / bounds the returned control input $\mathbf{u}$ to $[\mathbf{u}_{min}, \mathbf{u}_{max}]$.

The saturation could be helpful in case of soft constraints, that are violated for the planning and prediction, but that should be satisfied for the real system.

Specify the bounds via activateControlBounds().

Todo Support multiple control input bounds (now its scalar for all elements of $\mathbf{u}$).

Returns

Saturated control input u_k [q x 1].

Definition at line 168 of file teb_controller.h.

References teb::TebController< p, q >::_active_control_bounds, and teb::TebController< p, q >::firstControl().

16.39.4.15 template<int p, int q> virtual Eigen::VectorXd teb::TebController< p, q >::firstState ( ) const [inline], [virtual]

Returns

Copy of the first state

Implements teb::BaseController.

Definition at line 130 of file teb_controller.h.

References teb::TebController< p, q >::_state_seq.

16.39.4.16 template<int p, int q> const StateVector& teb::TebController< p, q >::firstStateRef ( ) const [inline]

Returns

Reference to the first state (read-only)

Definition at line 135 of file teb_controller.h.

References teb::TebController< p, q >::_state_seq.

16.39.4.17 template<int p, int q> Eigen::VectorXd teb::TebController< p, q >::getAbsoluteTimeVec ( ) const [virtual]

Calculate vector of absolute times $t = [0, \Delta T, 2\Delta T, \ldots, n\Delta T]$.

Returns

Row vector containing time information [n x 1].

Implements teb::BaseController.

Definition at line 780 of file teb_controller.hpp.

16.39.4.18 template<int p, int q> void teb::TebController< p, q >::getActiveVertices ( ) [protected]

Iterate through the TEB sequence/trajectory and determine if a vertex (StateVertex, ControlVertex or TimeDiff) is completely fixed.

In that case the corresponding vertex is not active and can be skipped during optimization (Jacobian calculation, Hessian, ...). All active vertices (pointers) are stored into TebController::_active_vertices.

While iterating all vertices, the hessian index is determined and stored inside the corresponding vertex (see TebVertex::setOptVecIdx() and TebVertex::getOptVecIdx()). For example:

- The final state (goal state) is fixed for many problems, in that case it does not appear in the optimization vector and therefore it is not part of the Hessian. But if at least a single component is unfixed, the complete vertex (TebSample in this case) becomes active. It's hessian or optimization vector index is zero. The number of free/unfixed variables is *q*.

- The second sample has no fixed components. It's hessian or optimization vector index is q since the last vertex starts at zero and has q free variables.
- The following vertex starts at $2 * q + p$ and so on ...
- The TimeDiff ΔT is stored at the end (and therefore it has the largest index if unfixed).

Todo Add case for different state and control input sequence lengths (e.g. for move blocking)

Definition at line 729 of file teb_controller.hpp.

16.39.4.19 template< int p, int q > virtual double teb::TebController< p, q >::getDt ( ) const [inline], [virtual]

Returns

ΔT

Implements teb::BaseController.

Definition at line 196 of file teb_controller.h.

References teb::TebController< p, q >::_dt, and teb::TimeDiff::dt().

16.39.4.20 template< int p, int q > virtual unsigned int teb::TebController< p, q >::getM ( ) const [inline], [virtual]

Returns

number of control vectors.

Definition at line 206 of file teb_controller.h.

References teb::TebController< p, q >::_ctrl_seq.

Referenced by teb::TebController< p, q >::returnControlInputSequence().

16.39.4.21 template< int p, int q > virtual unsigned int teb::TebController< p, q >::getN ( ) const [inline], [virtual]

Returns

number of state vectors.

Implements teb::BaseController.

Definition at line 201 of file teb_controller.h.

References teb::TebController< p, q >::_state_seq.

16.39.4.22 template< int p, int q > Eigen::Matrix< double,-1, 2 > teb::TebController< p, q >::getSampleOptVecIdxVMat ( ) const

Get info matrix that contains the Hessian index (first column) and the number of free variables (second column) for all states and control inputs (row-wise).

This function is used for debugging purposes, e.g. to test getActiveVertices().

Remarks

The Hessian index correspond to the row/col index of the first component of the current VertexType (StateVertex and ControlVertex). With the number of free variables, the position of the current VertexType inside the "big" composed Hessian is determined completely.

Returns

a copy of the hessian-idx-info-matrix [n x 2]

Definition at line 536 of file teb_controller.hpp.

16.39.4.23 template<int p, int q> Eigen::MatrixXd teb::TebController< p, q >::getStateCtrlInfoMat ( ) const [virtual]

Convert TEB to a full (p+q by n) matrix.

The new matrix is composed as follows:

- size: [p+q x n] (double), n=getN()
- rows 1:p -> states
- rows p+1:p+q -> control inputs
- columns 1:n -> discrete samples

Returns

new copy of the TEB matrix [p+q x n]

Implements teb::BaseController.

Definition at line 479 of file teb_controller.hpp.

16.39.4.24 template<int p, int q> Eigen::Matrix< bool, p+q,-1 > teb::TebController< p, q >::getTEBFixedMap ( ) const

Show fixed states of the TEB as a (p+q by n) matrix for debugging purposes.

The matrix is composed as follows:

- size: [p+q x n] (double)
- rows 1:p -> states
- rows p+1:p+q -> control inputs
- columns 1:n -> samples

Returns

a copy of the fixed state TEB info matrix [p+q x n]

Definition at line 508 of file teb_controller.hpp.

16.39.4.25 template<int p, int q> const **HyperGraph& teb::TebController**< p, q >::graph ( ) const [inline]

The hyper-graph stores all active vertices and edges (objectives and constraints) and is passed to the dedicated solver class. The hyper-graph is created within buildOptimizationGraph(). Users can add custom edges by subclassing and overriding customOptimizationGraph(). All active vertices are collected within getActiveVertices().

Returns

Read-only reference to the hyper-graph instance.

Definition at line 236 of file teb_controller.h.

References teb::TebController< p, q >::_graph.

16.39.4.26 template<int p, int q> void **teb::TebController**< p, q >::initOptimization ( )

Do everything what is necessary before calling the actual optimization optimizeTEB().

This wrapper function calls buildOptimizationGraph() to construct the hyper-graph. Afterwards it collects all active vertices using getActiveVertices().

In addition a few undesired config settings are checked (only if compiled in Debug mode).

Definition at line 689 of file teb_controller.hpp.

References teb::CENTRAL, and PRINT_DEBUG_COND_ONCE.

16.39.4.27 template<int p, int q> void **teb::TebController**< p, q >::initTrajectory (const Eigen::Ref< const **StateVector** > & *x0*, const Eigen::Ref< const **StateVector** > & *xf*, unsigned int *n*) [virtual]

This basic implementation linearly interpolates the trajectory between start $\mathbf{x}_0$ and final state $\mathbf{x}_f$.

Control inputs $\mathbf{u}_i$ are intialized to zero. This method also updates the number of discrete samples (state and control input vector pairs) *n*.

All existing states and control inputs of the sequences TebController::_state_seq and TebController::_ctrl_seq are deleted first.

Warning

Be careful, if you override this function in a subclass make sure to call HyperGraph::notifyGraphModified() inside.

Parameters

x0	copy of the current start state vector [p x 1]
xf	current final state vector [p x 1]
n	number of discrete samples (state and control input vector pairs) between start and goal

Todo More efficient implementation that keeps previously allocated memory

Definition at line 22 of file teb_controller.hpp.

16.39.4.28 template<int p, int q> void **teb::TebController**< p, q >::insertStateControlInputPair (unsigned int *sample_idx*, const Eigen::Ref< const **StateVector** > & *state*, const Eigen::Ref< const **ControlVector** > & *control*, bool *predict_control* = `true`)

Insert a new discrete StateVector ControlInput pair to TebController::_state_seq and TebController::_ctrl_seq at position `sample_idx`.

Existing elements between `idx+1` and `n` (getN()) are shifted to the interval `idx+1` and `n+1`.

if `predict_u` is `true`, a new control input u at position `sample_idx` is predicted using the class method predictControl().

Warning

Be careful, if you override this function in a subclass make sure to call HyperGraph::notifyGraphModified() inside.

Parameters

sample_idx	position at which the StateVertex and ControlVertex should be inserted
state	State vector [p x 1] to be added to _state_seq
control	Control input vector [q x 1] to be added _ctrl_seq
predict_control	if `true`, the control is not taken from `sample`, but it is calculated using the method predictControl() [default= `true`];

Definition at line 400 of file teb_controller.hpp.

16.39.4.29 template<int p, int q> virtual Eigen::VectorXd teb::TebController< p, q >::lastState ( ) const `[inline]`, `[virtual]`

Returns

Copy of the last state

Implements teb::BaseController.

Definition at line 140 of file teb_controller.h.

References teb::TebController< p, q >::_state_seq.

16.39.4.30 template<int p, int q> const StateVector& teb::TebController< p, q >::lastStateRef ( ) const `[inline]`

Returns

Reference to the last state (read-only)

Definition at line 145 of file teb_controller.h.

References teb::TebController< p, q >::_state_seq.

16.39.4.31 template<int p, int q> void teb::TebController< p, q >::optimizeTEB ( )

Start the actual TEB optimization routine.

The outer-loop resizes the trajectory (refer to resizeTrajectory() for infos), initializes the optimization problem (e.g. by creating the hyper-graph), and it calls the BaseSolver::solve() routine of the selected solver. The number of outer-loop iterations can be set in Config::Teb::teb_iter.

Todo The current implementation is really inefficient, since in each outer-loop-iteration all hyper-graph edges are deleted and afterwards reinstantiated. Create a dedicated update function in the future to hot-start from a previous hyper-graph.

See Also

initOptimization(), resizeTrajectory(), BaseSolver::solve(), setSolver()

Definition at line 312 of file teb_controller.hpp.

Referenced by teb::TebController< p, q >::step().

16.39.4.32 template< int p, int q > void **teb::TebController**< p, q >::predictControl (Eigen::Ref< **ControlVector** > *ctrl_out*, const Eigen::Ref< const **StateVector** > & *x1*, const Eigen::Ref< const **StateVector** > & *x2*) `[inline]`

This method can be used to predict the plant input $\mathbf{u}_i$ that transite drive the system from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$.

The method can be used e.g. in the trajectory initialization phase, or inseration/deletion of new discrete samples.

The current implementation just copies the previous prediction (which is set to 0 inside the method initTrajectory()). Problem specific subclasses can override this function and can use the inverse of the discrete system dynamics to derive a function for $\mathbf{u}_i$ with respect $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$.

E.g. for linear scalar systems:

$$\dot{x} = Ax + bu \tag{16.1}$$

$$x_{k+1} = x_k + \Delta T A x_k + \Delta T b u_k \tag{16.2}$$

$$u_k = \frac{x_{k+1} - x_k - \Delta T A}{\Delta T b} \tag{16.3}$$

Parameters

ctrl_out	[output] predicted control input vector $\mathbf{u}_i$ [q x 1]
x1	vector state $\mathbf{x}_i$ at discrete time *i* [p x 1]
x2	vector state $\mathbf{x}_{i+1}$ at discrete time *i+1* [p x 1]

Definition at line 462 of file teb_controller.hpp.

16.39.4.33 template< int p, int q > void **teb::TebController**< p, q >::pushBackStateControlInputPair (const Eigen::Ref< const **StateVector** > & *state*, const Eigen::Ref< const **ControlVector** > & *control*) `[inline]`

Parameters

state	State vector [p x 1] to be added to _state_seq
control	Control input vector [q x 1] to be added _ctrl_seq

Definition at line 253 of file teb_controller.h.

References teb::TebController< p, q >::_ctrl_seq, and teb::TebController< p, q >::_state_seq.

16.39.4.34 template< int p, int q > void **teb::TebController**< p, q >::pushFrontStateControlInputPair (const Eigen::Ref< const **StateVector** > & *state*, const Eigen::Ref< const **ControlVector** > & *control*) `[inline]`

Parameters

state	State vector [p x 1] to be added to _state_seq
control	Control input vector [q x 1] to be added _ctrl_seq

Definition at line 264 of file teb_controller.h.

References teb::TebController< p, q >::_ctrl_seq, and teb::TebController< p, q >::_state_seq.

16.39.4.35 template< int p, int q > void **teb::TebController**< p, q >::removeStateControlInputPair (unsigned int *sample_idx*, bool *predict_control* = `true`)

Remove an existing discrete state control input pair from the object properties TebController::_state_seq and TebController::_ctrl_seq at position `sample_idx`.

Existing elements between `sample_idx+1` and `n` (getN()) are shifted to the interval `idx` and `n-1`.

If `predict_control` is `true` A new control input u at position `sample_idx` is predicted using the class method predictControl().

Warning

Be careful, if you override this function in a subclass make sure to call HyperGraph::notifyGraphModified() inside.

Parameters

sample_idx	index in TebController::_state_seq and TebController::_ctrl_seq to be deleted
predict_control	if `true`, the control for the new interval is calculated using the method predictControl() [default=`true`];

Definition at line 351 of file teb_controller.hpp.

16.39.4.36 template<int p, int q> void teb::TebController< p, q >::resampleTrajectory (unsigned int *n_new*)

This function changes the number of samples by resampling the complete trajectories TebController::_state_seq and TebController::_ctrl_seq ($\mathbf{x}_i$, $\mathbf{u}_i$, ΔT).

Currently only linear interpolation is implemented.

This function may called withing resizeTrajectory() that is part of the TEB optimization loop (see optimizeTEB()).

To add/remove a single state refer to insertSample() / removeSample().

Warning

Be careful, if you override this function in a subclass make sure to call HyperGraph::notifyGraphModified() inside.

Parameters

n_new	new number of samples

Todo More efficient strategy without copying the complete sequences.

@ todo If someone freezes states within the sequence (in addition to start and goal), it is not set to unfreezed here... But we do not want to force all states to be unfreezed due to lack of efficiency...

Todo : later this should be n_new-1

Definition at line 120 of file teb_controller.hpp.

16.39.4.37 template<int p, int q> virtual void teb::TebController< p, q >::resetController ( ) `[inline]`, `[virtual]`

Implements teb::BaseController.

Definition at line 357 of file teb_controller.h.

References teb::TebController< p, q >::_ctrl_seq, teb::TebController< p, q >::_dt, teb::TebController< p, q >::_dt_ref, teb::TebController< p, q >::_graph, teb::TebController< p, q >::_state_seq, teb::TebController< p, q >::cfg, teb::HyperGraph::clearGraph(), teb::TimeDiff::dt(), teb::Config::Teb::dt_ref, and teb::Config::teb.

16.39.4.38 template<int p, int q> void teb::TebController< p, q >::resizeTrajectory ( )

Resizing the trajectory is helpful e.g.

for the following scenarios:

- Large disturbances or obstacles requires the teb to be extended in order to satisfy the given sample rate and avoid undesirable behavior due to a large/small discretization step width ΔT. After clearance of disturbances/obstacles the teb should contract to its (time-)optimal trajectory again.
- If the distance to the goal state is getting smaller, dt is decreasing as well. This leads to a heavily fine-grained discretization in combination with many discrete samples. Thus, the computation time will be/remain high and in addition numerical instabilities can appear (e.g. due to the division by a small ΔT).

The implemented stragety observes the timediff ΔT. and

- inserts a new sample if $\Delta T > \Delta T_{ref} + \Delta T_{hyst}$
- removes a sample if $\Delta T < \Delta T_{ref} - \Delta T_{hyst}$

ΔT_{ref} and ΔT_{hyst} can be changed in Config::teb.

Since the ΔT is currently chosen uniformly, no information about a possible position (at which the new sample should be inserted) can be obtained. This function resamples linearly the whole trajectory using resampleTrajectory().

If the time difference vertex TimeDiff is fixed during optimization (TimeDiff::_fixed == `true` , TimeDiff::isFixedAll()), this method returns without any changes.

Definition at line 86 of file teb_controller.hpp.

16.39.4.39 template< int p, int q> virtual Eigen::MatrixXd teb::TebController< p, q >::returnControlInputSequence ( ) const [inline], [virtual]

Returns

Matrix containing all planned control inputs [q x n-1]

Todo Here we assume, that the last control input is invalid - We need to change this, if we have different sizes for state and control input sequences

Implements teb::BaseController.

Definition at line 211 of file teb_controller.h.

References teb::TebController< p, q >::_ctrl_seq, and teb::TebController< p, q >::getM().

16.39.4.40 template< int p, int q> void teb::TebController< p, q >::setFixedDt (bool *fixed*) [inline]

See Also

setupHorizon(), setGoalStatesFixedOrUnfixed()

Parameters

fixed	Set to `true` in order to fix TebController::_dt.

Definition at line 583 of file teb_controller.h.

References teb::TebController< p, q >::_dt, teb::TebController< p, q >::_graph, teb::HyperGraph::notifyGraphModified(), and teb::TimeDiff::setFixedAll().

Referenced by teb::TebController< p, q >::setupHorizon().

16.39.4.41 template< int p, int q> void teb::TebController< p, q >::setFixedGoal (bool *fixed*) [inline], [protected]

Parameters

fixed	Set to `true` in order to fix the states.

Definition at line 672 of file teb_controller.h.

References teb::TebController< p, q >::_graph, teb::TebController< p, q >::_state_seq, and teb::HyperGraph::notifyGraphModified().

16.39.4.42 template<int p, int q> void **teb::TebController**< p, q >::setFixedGoal (const bool * *fixed*) [inline], [protected]

Parameters

fixed	Boolean array with *p* components. Set i-th component to `true` in order to fix the corresponding state.

Definition at line 684 of file teb_controller.h.

References teb::TebController< p, q >::_graph, teb::TebController< p, q >::_state_seq, and teb::HyperGraph::notifyGraphModified().

16.39.4.43 template<int p, int q> void **teb::TebController**< p, q >::setFixedGoal (const Eigen::Ref< const Eigen::Matrix< bool, p, 1 >> & *fixed*) [inline], [protected]

Parameters

fixed	Eigen::Vector with *p* booleans. Set i-th component to `true` in order to fix the corresponding state.

Definition at line 696 of file teb_controller.h.

References teb::TebController< p, q >::_graph, teb::TebController< p, q >::_state_seq, and teb::HyperGraph::notifyGraphModified().

16.39.4.44 template<int p, int q> void **teb::TebController**< p, q >::setFixedStart (bool *fixed*) [inline], [protected]

Parameters

fixed	Set to `true` in order to fix the states.

Definition at line 648 of file teb_controller.h.

References teb::TebController< p, q >::_graph, teb::TebController< p, q >::_state_seq, and teb::HyperGraph::notifyGraphModified().

16.39.4.45 template<int p, int q> void **teb::TebController**< p, q >::setFixedStart (const bool * *fixed*) [inline], [protected]

Parameters

fixed	Boolean array with *p* components. Set i-th component to `true` in order to fix the corresponding state.

Definition at line 660 of file teb_controller.h.

References teb::TebController< p, q >::_graph, teb::TebController< p, q >::_state_seq, and teb::HyperGraph::notifyGraphModified().

16.39.4.46 template<int p, int q> void teb::TebController< p, q >::setGoalStatesFixedOrUnfixed (const bool * *fixed_goal_states*) [inline]

Parameters

fixed_goal_-states	boolean array of size `p`. Each component of lastState() is set to fixed() if `fixed_goal_-states`[i] == `true`.

Definition at line 273 of file teb_controller.h.

References teb::TebController< p, q >::_fixed_goal_states, teb::TebController< p, q >::_graph, and teb::Hyper-Graph::notifyGraphModified().

Referenced by teb::TebController< p, q >::setupHorizon().

16.39.4.47 template<int p, int q> void **teb::TebController**< p, q >::setGoalStatesFixedOrUnfixed (const Eigen::Ref< const Eigen::Matrix< bool, p, 1 >> & *fixed_goal_states*) [inline]

Parameters

fixed_goal_-states	Eigen::Vector of size `p` containing booleans. Each component of lastState() is set to fixed() if `fixed_goal_states`[i] == `true`.

Definition at line 283 of file teb_controller.h.

References teb::TebController< p, q >::_fixed_goal_states, teb::TebController< p, q >::_graph, and teb::Hyper-Graph::notifyGraphModified().

16.39.4.48 template<int p, int q> void **teb::TebController**< p, q >::setGoalStatesFixedOrUnfixed (bool *all_goal_states_fixed*) [inline]

See Also

setupHorizon(), setFixedDt()

Parameters

all_goal_states_-fixed	if `true`, all components of lastState() are set to fixed()

Definition at line 294 of file teb_controller.h.

References teb::TebController< p, q >::_fixed_goal_states, teb::TebController< p, q >::_graph, and teb::Hyper-Graph::notifyGraphModified().

16.39.4.49 template<int p, int q> void **teb::TebController**< p, q >::setSolver (**BaseSolver** ∗ *solver*) [inline]

Set subclasses of BaseSolver here. Solver classes are frequently expanded.

Remarks

Subclasses of BaseSolverLeastSquares are least-squares solvers that are efficient, but originally not suited for hard constraints. In addition, the objective function must be a quadratic function. In order to allow similar functionalities, the cost/objective function is squared $\min V^2(\mathscr{B})$ and equality/inequality constraints are approximated by quadratic soft-constraints [5].

Parameters

solver	Pointer to subclass of BaseSolver. TebController does not free memory of the solver class.

See Also

BaseSolver BaseSolverLeastSquares

Definition at line 626 of file teb_controller.h.

References teb::TebController< p, q >::_solver, teb::TebController< p, q >::cfg, and teb::BaseSolver::setConfig().

Referenced by teb::TebController< p, q >::TebController().

16.39.4.50 template<int p, int q> void **teb::TebController**< p, q >::setSystemDynamics (**SystemDynamics**< p, q > * *system*, bool *activate* = `true`) [inline]

Add system dynamics equations to the optimization problem. System equations are defined as a equality constraints since they should be satisfied all the time.

See SystemDynamics class for more details.

Parameters

system	pointer to the SystemDynamics (sub-)class that stores the system dynamics equations.
activate	if set to `true` than the edge is activated.

See Also

buildOptimizationGraph(), activateObjectiveTimeOptimal(), activateStateBounds(), activateControlBounds()

Definition at line 604 of file teb_controller.h.

References teb::TebController< p, q >::_active_system_dynamics, teb::TebController< p, q >::_graph, and teb::-HyperGraph::notifyGraphModified().

16.39.4.51 template<int p, int q> void **teb::TebController**< p, q >::setupHorizon (bool *fixed_goal*, bool *fixed_resolution*, int *no_samples* = -1, double *dt* = -1) [inline]

Use this function to setup different common horizon settings at once instead of changing config parameters and class members individually.

Common horizons:

- Receding/moving horizon: unfixed goal and fixed resolution of samples (fixed dt)
- Fixed end-point: fixed goal and fixed resolution of samples (fixed dt)
- TEB default: fixed goal and unfixed resolution (unfixed dt, in order to minimize time)

If you choose an unfixed resoltion (`fixed_resolution == false`) make sure that there are objectives considering the time difference ΔT, otherwise the problam will probably be ill-posed.

If you want to fix or unfix only a subset of goal states, use the method setGoalStatesFixedOrUnfixed() after calling setupHorizon or without calling setupHorizon.

See Also

setFixedDt(), setGoalStatesFixedOrUnfixed()

Parameters

fixed_goal	Set to `true` in order to fix or unfix all goal state variables.
fixed_resolution	Set to `true` in order to fix the resolution (the time difference ΔT between consecutive samples).
no_samples	Set default/initial number of samples (it remains constant, if the resolution is fixed as well). If -1, Config::Teb::n_pre is used.
dt	Set reference time difference for discretization (it remains constant, if the resolution is fixed as well). If -1, Config::Teb::dt_ref is used.

Definition at line 327 of file teb_controller.h.

References teb::TebController< p, q >::_dt, teb::TebController< p, q >::_dt_ref, teb::TebController< p, q >::_no_samples, teb::TebController< p, q >::cfg, teb::TebController< p, q >::dt(), teb::TimeDiff::dt(), teb::Config::Teb::dt_min, teb::Config::Teb::dt_ref, teb::Config::Teb::n_min, teb::Config::Teb::n_pre, teb::TebController< p, q >::setFixedDt(), teb::TebController< p, q >::setGoalStatesFixedOrUnfixed(), and teb::Config::teb.

16.39.4.52 template<int p, int q> const StateSequence& teb::TebController< p, q >::stateSequence ( ) const [inline]

Returns

Reference to the state sequence (read-only)

Definition at line 181 of file teb_controller.h.

References teb::TebController< p, q >::_state_seq.

16.39.4.53 template<int p, int q> virtual void teb::TebController< p, q >::step (const double ∗const *x0*, const double ∗const *xf*, double ∗ *ctrl_out* = `nullptr`) [inline], [virtual]

Parameters

x0	double array containing start state `x0` with `p` components [`p` x 1]
xf	double array containing final state `xf` with `p` components [`p` x 1]
ctrl_out	[output] store control input (firstControl()) to control the plant [`q` x 1]

Implements teb::BaseController.

Definition at line 398 of file teb_controller.h.

16.39.4.54 template<int p, int q> void teb::TebController< p, q >::step (const Eigen::Ref< const StateVector > & *x0*, const Eigen::Ref< const StateVector > & *xf*, Eigen::Map< ControlVector > ∗ *ctrl_out* = `nullptr`) [inline]

Calls updateStart(), updateGoal(), optimizeTEB() and firstControl().

Parameters

x0	StateVector containing start state `x0` with `p` components [`p` x 1]
xf	StateVector containing final state `xf` with `p` components [`p` x 1]
ctrl_out	[output] store control input (firstControl()) to ControlVector-Map [`q` x 1]

Definition at line 412 of file teb_controller.h.

References teb::TebController< p, q >::firstControl(), teb::TebController< p, q >::optimizeTEB(), teb::TebController< p, q >::updateGoal(), and teb::TebController< p, q >::updateStart().

16.39.4.55 template<int p, int q> void teb::TebController< p, q >::updateGoal (const Eigen::Ref< const StateVector > & *xf*) [virtual]

Update goal/final StateVector.

If getN()==1 (that is if only start state exists after setting updateStart()) than initTrajectory() is called to initialize a trajectory between previously defiend start and xf.

If an existing goal (lastState()) is far away (Config::Teb::goal_dist_force_reinit), the trajectory is reinitialized using initTrajectory() as well.

If the goal is not completely fixed during optimization, only unfixed components of the final state are updated.

Warning

Be careful, if you override this function in a subclass make sure to call HyperGraph::notifyGraphModified() inside.

Parameters

xf	new final state vector [p x 1]

Definition at line 261 of file teb_controller.hpp.

Referenced by teb::TebController< p, q >::step().

16.39.4.56 template<int p, int q> void **teb::TebController**< p, q >::updateStart (const Eigen::Ref< const **StateVector** > & *x0*) [virtual]

Update first/start StateVector (start of the trajectory) with a new one (x_start).

This can be obtained from the most recent measurements. The teb vector is pruned if a state with idx i>0 is found, that has a smaller distance (l2-norm) to the new x0 in comparance to firstState().

If the TEB vector is empty, the start state is added anyway (as single state trajectory).

Warning

Be careful, if you override this function in a subclass make sure to call HyperGraph::notifyGraphModified() inside.

Parameters

x0	new start state vector [p x 1]

See Also

updateGoal(), initTrajectory()

Definition at line 203 of file teb_controller.hpp.

Referenced by teb::TebController< p, q >::step().

16.39.5 Member Data Documentation

16.39.5.1 template<int p, int q> std::tuple<bool, **ControlVector**, **ControlVector**> **teb::TebController**< p, q >::_active_control_bounds = std::make_tuple(false,ControlVector::Constant(INF),ControlVector::Constant(-INF)) [protected]

- First type (`bool`): if `true`, EdgeControlBounds is activated.
- Second type (`ControlVector`): lower bound on each ControlVector u_{min} (component-wise).
- Third type (`ControlVector`): upper bound on each ControlVector u_{max} (component-wise).

See Also

EdgeControlBounds, buildOptimizationGraph()

Definition at line 770 of file teb_controller.h.

Referenced by teb::TebController< p, q >::activateControlBounds(), and teb::TebController< p, q >::firstControl-Saturated().

16.39.5.2 template<int p, int q> std::tuple<bool, double, double, double> teb::TebController< p, q >::_active_quadratic_form = std::make_tuple(false,1,1,1) [protected]

- First type (`bool`): if `true`, EdgeQuadraticForm is activated (quadratic cost on states and control inputs).
- Second type (`Weight` Q): Weight for the state vectors (uniform weight here)
- Third type (`Weight` R): Weight for the control input vectors (uniform weight here)
- Fourth type (`Weight` Qf): Weight for the final state vector (uniform weight here)

See Also

activateObjectiveQuadraticForm(), EdgeControlBounds, buildOptimizationGraph()

Definition at line 752 of file teb_controller.h.

Referenced by teb::TebController< p, q >::activateObjectiveQuadraticForm().

16.39.5.3 template<int p, int q> std::tuple<bool, StateVector, StateVector> teb::TebController< p, q >::_active_state_bounds = std::make_tuple(false, StateVector::Constant(-INF), StateVector::Constant(INF)) [protected]

- First type (`bool`): if `true`, EdgeStateBounds is activated.
- Second type (`StateVector`): lower bound on each StateVector x_{min} (component-wise).
- Third type (`StateVector`): upper bound on each StateVector x_{max} (component-wise).

See Also

EdgeStateBounds, buildOptimizationGraph()

Definition at line 779 of file teb_controller.h.

Referenced by teb::TebController< p, q >::activateStateBounds().

16.39.5.4 template<int p, int q> std::pair<bool,SystemDynamics<p, q>*> teb::TebController< p, q >::_active_system_dynamics = std::make_pair(false,nullptr) [protected]

- First type (`bool`): if `true`, EdgeSystemDynamics is activated.
- Second type (`SystemDynamics`): pointer to SystemDynamics object.

See Also

setSystemDynamics(), SystemDynamics, EdgeSystemDynamics, buildOptimizationGraph()

Definition at line 761 of file teb_controller.h.

Referenced by teb::TebController< p, q >::setSystemDynamics().

16.39.5.5 template< int p, int q > bool **teb::TebController**< p, q >::_active_time_optimal = false [protected]

Definition at line 742 of file teb_controller.h.

Referenced by teb::TebController< p, q >::activateObjectiveTimeOptimal().

16.39.5.6 template< int p, int q > bool **teb::TebController**< p, q >::_cfg_owned = false [protected]

Definition at line 737 of file teb_controller.h.

Referenced by teb::TebController< p, q >::TebController(), and teb::TebController< p, q >::~TebController().

16.39.5.7 template< int p, int q > **ControlSequence teb::TebController**< p, q >::_ctrl_seq [protected]

Definition at line 712 of file teb_controller.h.

Referenced by teb::TebController< p, q >::controlSequence(), teb::TebController< p, q >::firstControl(), teb::TebController< p, q >::firstControlRef(), teb::TebController< p, q >::getM(), teb::TebController< p, q >::pushBackStateControlInputPair(), teb::TebController< p, q >::pushFrontStateControlInputPair(), teb::TebController< p, q >::resetController(), and teb::TebController< p, q >::returnControlInputSequence().

16.39.5.8 template< int p, int q > **TimeDiff teb::TebController**< p, q >::_dt [protected]

Stores ΔT. Another interpretation: discretization step width. This implementation supports currently a uniformly distributed dt only.

Definition at line 720 of file teb_controller.h.

Referenced by teb::TebController< p, q >::dt(), teb::TebController< p, q >::getDt(), teb::TebController< p, q >::resetController(), teb::TebController< p, q >::setFixedDt(), and teb::TebController< p, q >::setupHorizon().

16.39.5.9 template< int p, int q > double **teb::TebController**< p, q >::_dt_ref = -1 [protected]

Definition at line 726 of file teb_controller.h.

Referenced by teb::TebController< p, q >::resetController(), and teb::TebController< p, q >::setupHorizon().

16.39.5.10 template< int p, int q > **FixedStates teb::TebController**< p, q >::_fixed_goal_states = FixedStates::Constant(true) [protected]

Definition at line 733 of file teb_controller.h.

Referenced by teb::TebController< p, q >::setGoalStatesFixedOrUnfixed().

16.39.5.11 template< int p, int q > **StateVector teb::TebController**< p, q >::_goal_backup [protected]

Definition at line 723 of file teb_controller.h.

16.39.5.12 template< int p, int q > **HyperGraph teb::TebController**< p, q >::_graph [protected]

Definition at line 731 of file teb_controller.h.

Referenced by teb::TebController< p, q >::activateControlBounds(), teb::TebController< p, q >::activateObjectiveQuadraticForm(), teb::TebController< p, q >::activateObjectiveTimeOptimal(), teb::TebController< p, q >::activateStateBounds(), teb::TebController< p, q >::graph(), teb::TebController< p, q >::resetController(), teb::TebController< p, q >::setFixedDt(), teb::TebController< p, q >::setFixedGoal(), teb::TebController< p, q

>::setFixedStart(), teb::TebController< p, q >::setGoalStatesFixedOrUnfixed(), teb::TebController< p, q >::setSystemDynamics(), and teb::TebController< p, q >::~TebController().

16.39.5.13 template<int p, int q> int teb::TebController< p, q >::_no_samples = -1 [protected]

Definition at line 725 of file teb_controller.h.

Referenced by teb::TebController< p, q >::setupHorizon().

16.39.5.14 template<int p, int q> bool teb::TebController< p, q >::_optimized [protected]

Definition at line 729 of file teb_controller.h.

16.39.5.15 template<int p, int q> BaseSolver* teb::TebController< p, q >::_solver = nullptr [protected]

Definition at line 735 of file teb_controller.h.

Referenced by teb::TebController< p, q >::setSolver().

16.39.5.16 template<int p, int q> StateSequence teb::TebController< p, q >::_state_seq [protected]

Definition at line 711 of file teb_controller.h.

Referenced by teb::TebController< p, q >::firstState(), teb::TebController< p, q >::firstStateRef(), teb::TebController< p, q >::getN(), teb::TebController< p, q >::lastState(), teb::TebController< p, q >::lastStateRef(), teb::TebController< p, q >::pushBackStateControlInputPair(), teb::TebController< p, q >::pushFrontStateControlInputPair(), teb::TebController< p, q >::resetController(), teb::TebController< p, q >::setFixedGoal(), teb::TebController< p, q >::setFixedStart(), and teb::TebController< p, q >::stateSequence().

16.39.5.17 template<int p, int q> const Config* teb::TebController< p, q >::cfg

If the Config class is provided via TebController::TebController() than the memory management is not touched by this class. If no Config class is provided manually, memory management is owned by this class.

See Also

TebController::TebController(), TebController::_cfg_owned

Definition at line 90 of file teb_controller.h.

Referenced by teb::TebController< p, q >::resetController(), teb::TebController< p, q >::setSolver(), teb::TebController< p, q >::setupHorizon(), and teb::TebController< p, q >::~TebController().

16.39.5.18 template<int p, int q> const unsigned int teb::TebController< p, q >::NoControls = q [static]

Definition at line 60 of file teb_controller.h.

16.39.5.19 template<int p, int q> const unsigned int teb::TebController< p, q >::NoStates = p [static]

Definition at line 59 of file teb_controller.h.

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

- teb_controller.h
- teb_controller.hpp

16.40 teb::TebPlotter Class Reference

Provides useful visualization and plotting functions.

`#include <teb_plotter.h>`

Public Types

- enum FileFormat {
  FileFormat::PNG, FileFormat::JPEG, FileFormat::SVG, FileFormat::PDF,
  FileFormat::EPS, FileFormat::TIKZ }

  Supported file formats for figure export.

Public Member Functions

- TebPlotter ()

  Constructor.
- ∼TebPlotter ()

  Destructor.
- void plot1DVector (const Eigen::Ref< const Eigen::VectorXd > &data)

  Plot a single vector over the range 0 .. n.
- void plot (const Eigen::Ref< const Eigen::VectorXd > &x, const Eigen::Ref< const Eigen::VectorXd > &y, std::string title="", std::string xlabel="", std::string ylabel="")

  Create a common 2D plot.
- void spyMatrix (const Eigen::Ref< const Eigen::MatrixXd > &matrix)

  Plot the sparsity pattern of a given matrix.
- template<typename StateSeq , typename ControlSeq >
  void plotTEB (const StateSeq &state_seq, ControlSeq &ctrl_seq, double dt)

  Plot TEB states and control inputs.
- template<typename TebCtrl >
  void plotTEB (const TebCtrl &teb)

  Plot TEB states and control inputs.
- void plotTwoCol (const Eigen::Ref< const Eigen::VectorXd > &col1_time, const Eigen::Ref< const Eigen::MatrixXd > &col1_data, const Eigen::Ref< const Eigen::VectorXd > &col2_time, const Eigen::Ref< const Eigen::MatrixXd > &col2_data, PlotOptions ∗options=nullptr)

  Create a two column multiplot with a single line for each plot.
- void plotTwoCol (const std::vector< const Eigen::VectorXd ∗ > &col1_time, const std::vector< const Eigen::MatrixXd ∗ > &col1_data, const std::vector< const Eigen::VectorXd ∗ > &col2_time, const std::vector< const Eigen::MatrixXd ∗ > &col2_data, PlotOptions ∗options=nullptr)

  Create a two column multiplot with multiple lines for each plot.
- void plotMulti (const Eigen::Ref< const Eigen::VectorXd > &col1_time, const Eigen::Ref< const Eigen::MatrixXd > &col1_data, PlotOptions ∗options=nullptr)

  Create a one column multiplot with a single line for each plot.
- void plotMulti (const std::vector< const Eigen::VectorXd ∗ > &col1_time, const std::vector< const Eigen::MatrixXd ∗ > &col1_data, PlotOptions ∗options=nullptr)

  Create a one column multiplot with multiple lines for each plot.
- void clearWindow ()

  Clear current gnuplot window.
- void switchWindow (int window_id, bool reset=true)

  Switch gnuplot window or create a new one.
- void closeWindow (int window_id=0)

  Close gnuplot window with id `window_id`.

- void setOutputToFile (std::string filename, FileFormat format=FileFormat::PNG)

  Change output to file.
- void completePdfExport ()

  Use this function to complete the pdf export after plotting.
- void setOutputToWindow ()

  Change output to a common gnuplot window.
- bool isExportToFileEnabled ()

  Query status if export to file is enabled.

Protected Member Functions

- void plotCustomKey (const std::vector< std::string > &keys)

  Plot a legend into a multiplot environment.

Protected Attributes

- bool _pdf_flag = false

  `true` *if PDF export is enabled.*
- bool _file_export = false

  `true` *if export to file is desired*
- FILE * pipe = nullptr

16.40.1 Detailed Description

This class provides visualization and plot functions for dynamic systems and the TebController. In addition this class defines plot functions for custom data specified by Eigen::Vector types.

Requirements: This class needs gnuplot installed correctly and it must be determined by cmake first (GNUPLOT_-PATH needs to be valid).

Bug EPS and PDF export are not working as expected. No text is displayed and after importing to Inskape, the plot is blank (noticed on Ubuntu 14.04).

Author

Christoph Rösmann (`christoph.roesmann@tu-dortmund.de`)

Examples:

integrator_system1.cpp, integrator_system2.cpp, integrator_system_classic_mpc.cpp, linear_system_ode.-cpp, linear_system_ode_ctrl_comparison.cpp, linear_system_state_space.cpp, mobile_robot_teb.cpp, rocket-_system.cpp, and van_der_pol_system.cpp.

Definition at line 56 of file teb_plotter.h.

16.40.2 Member Enumeration Documentation

16.40.2.1 enum teb::TebPlotter::FileFormat [strong]

Enumerator

PNG

JPEG

SVG

PDF

EPS

TIKZ

Definition at line 61 of file teb_plotter.h.

16.40.3 Constructor & Destructor Documentation

16.40.3.1 teb::TebPlotter::TebPlotter ( )

Constructs a new TebPlotter object.

A new pipe to gnuplot is initialized. The default output is set to a new window.

Definition at line 13 of file teb_plotter.cpp.

References pipe, PRINT_INFO, and setOutputToWindow().

16.40.3.2 teb::TebPlotter::∼TebPlotter ( )

Destructs the TebPlotter object.

The pipe to gnuplot is closed.

Definition at line 38 of file teb_plotter.cpp.

References completePdfExport(), and pipe.

16.40.4 Member Function Documentation

16.40.4.1 void teb::TebPlotter::clearWindow ( ) [inline]

Definition at line 121 of file teb_plotter.h.

References pipe.

16.40.4.2 void teb::TebPlotter::closeWindow (int *window_id* = 0) [inline]

Parameters

window_id	that should be closed.

Definition at line 146 of file teb_plotter.h.

References pipe.

16.40.4.3 void teb::TebPlotter::completePdfExport ( ) [inline]

Definition at line 216 of file teb_plotter.h.

References _pdf_flag, and pipe.

Referenced by ∼TebPlotter().

16.40.4.4 bool teb::TebPlotter::isExportToFileEnabled ( ) [inline]

Returns

`true`, if export to file is enabled

Definition at line 243 of file teb_plotter.h.

References _file_export.

16.40.4.5 void teb::TebPlotter::plot (const Eigen::Ref< const Eigen::VectorXd > & *x*, const Eigen::Ref< const Eigen::VectorXd > & *y*, std::string *title* = "", std::string *xlabel* = "", std::string *ylabel* = "")

Create a simple 2D plot using two different n-dim vectors for x and y.

Parameters

x	x-data vector (abscissa): Eigen::Vector [n x 1]
y	y-data vector (ordinate): Eigen::Vector [n x 1]
title	String defining the title of the plot (optional)
xlabel	String defining the label of the x-axis (optional)
ylabel	String defining the label of the y-axis (optional)

disable legend

show grid

plot title

plot xlabel

plot ylabel

plot type

loop over the data

data terminated with

termination character

flush the pipe

Definition at line 81 of file teb_plotter.cpp.

References pipe, and PRINT_INFO.

16.40.4.6 void teb::TebPlotter::plot1DVector (const Eigen::Ref< const Eigen::VectorXd > & *data*)

Plot a single vector.

The abscissa is chosen uniformly to 0 .. n

Parameters

data	Y data wrapped into an Eigen::Vector [n x 1]

disable legend

show grid

plot type

loop over the data

data terminated with

termination character

flush the pipe

Definition at line 55 of file teb_plotter.cpp.

References pipe, and PRINT_INFO.

16.40.4.7 void teb::TebPlotter::plotCustomKey (const std::vector< std::string > & *keys*) [protected]

Use this function to plot a custom legend into a multicol environment.

\ Specify location before calling this function by setting gnuplot margins `tmargin`, `bmargin`, `lmargin` and `rmargin`.

Parameters

keys	Vector of legend strings (for each line)

WORKS ONLY IN MULTI PLOT ENVIRONMENT!

terminate

Definition at line 574 of file teb_plotter.cpp.

References pipe.

Referenced by plotTwoCol().

16.40.4.8 void teb::TebPlotter::plotMulti (const Eigen::Ref< const Eigen::VectorXd > & *col1_time*, const Eigen::Ref< const Eigen::MatrixXd > & *col1_data*, PlotOptions ∗ *options* = `nullptr`) [inline]

Definition at line 103 of file teb_plotter.h.

References plotTwoCol().

16.40.4.9 void teb::TebPlotter::plotMulti (const std::vector< const Eigen::VectorXd ∗ > & *col1_time*, const std::vector< const Eigen::MatrixXd ∗ > & *col1_data*, PlotOptions ∗ *options* = `nullptr`) [inline]

Definition at line 111 of file teb_plotter.h.

References plotTwoCol().

16.40.4.10 template<typename StateSeq , typename ControlSeq > void teb::TebPlotter::plotTEB (const StateSeq & *state_seq*, ControlSeq & *ctrl_seq*, double *dt*)

Use this function to plot the state and control input sequence of the TebController given as TebController::TEBVector type.

Remarks

Based on `http://stackoverflow.com/questions/14712251/place-key-below-multiplot-graph-i` and extended to two column multiplots

Parameters

state_seq	State sequence (Vectors of type StateVertex)
ctrl_seq	Ccontrol input sequence (Vectors of type ControlVertex)
dt	Time interval between two consecutive states and control inputs ΔT.

Template Parameters

TebVec	Type of the TebVec (deduced by the compiler).

Examples:

integrator_system1.cpp, linear_system_ode.cpp, linear_system_state_space.cpp, and van_der_pol_system.-cpp.

Definition at line 19 of file teb_plotter.hpp.

References pipe, and PRINT_INFO.

Referenced by plotTEB().

16.40.4.11 template< typename TebCtrl > void teb::TebPlotter::plotTEB (const TebCtrl & *teb*) [inline]

Parameters

teb	Reference to a TebController object

Template Parameters

Type	of the TebController object (deduced by the compiler)

Definition at line 87 of file teb_plotter.h.

References plotTEB().

16.40.4.12 void teb::TebPlotter::plotTwoCol (const Eigen::Ref< const Eigen::VectorXd > & *col1_time*, const Eigen::Ref< const Eigen::MatrixXd > & *col1_data*, const Eigen::Ref< const Eigen::VectorXd > & *col2_time*, const Eigen::Ref< const Eigen::MatrixXd > & *col2_data*, PlotOptions * *options* = `nullptr`)

The number of rows for each column may differ.

Remarks

No legend options implemented

Test What happens if the right column contains more rows than the left one? (Alignment)

Parameters

col1_time	time axis for the left column [n1 x 1]
col1_data	data for the left column: Each row corresponds to a single plot [rows1 x n1]
col2_time	time axis for the right column [n2 x 1]
col2_data	data for the right column: Each row corresponds to a single plot [rows2 x n2]
options	Pointer to customized plot options given by a PlotOptions object.

Definition at line 122 of file teb_plotter.cpp.

References pipe, PRINT_INFO, teb::PlotOptions::skip_last_value_right_column, teb::PlotOptions::title, and teb::-PlotOptions::ylabels.

Referenced by plotMulti().

16.40.4.13 void teb::TebPlotter::plotTwoCol (const std::vector< const Eigen::VectorXd * > & *col1_time*, const std::vector< const Eigen::MatrixXd * > & *col1_data*, const std::vector< const Eigen::VectorXd * > & *col2_time*, const std::vector< const Eigen::MatrixXd * > & *col2_data*, PlotOptions * *options* = `nullptr`)

The number of rows for each column may differ.

One element of the col1 arguments stores the complete information for all subplots. Other elements extend each plot by new lines. E.g:

- `col1_data.at(0)` contains 3 rows, that results 3 subplots / rows for the left column. Each row specifies one line for each plot.
- `col1_data.at(1)` contains 3 rows as well. Each line is added to its corresponding subplots.

  This requires the same number of samples (columns) between `col1_data.at(i)` and col1_time.at(i).

Parameters

col1_time	container of time data for the lines in the left column
col1_data	container of y-data for the lines in the left colum
col2_time	container of time data for the lines in the right column
col2_data	container of y-data for the lines in the right colum
options	Pointer to customized plot options given by a PlotOptions object.

Definition at line 276 of file teb_plotter.cpp.

References teb::PlotOptions::legend, teb::PlotOptions::legend_entries, pipe, plotCustomKey(), PRINT_INFO, teb::-PlotOptions::skip_last_value_right_column, teb::PlotOptions::title, and teb::PlotOptions::ylabels.

16.40.4.14 void teb::TebPlotter::setOutputToFile (std::string *filename,* FileFormat *format* = FileFormat::PNG) [inline]

Parameters

filename	Export figure to a file called filename into the current working directory or specify complete path.
format	Fileformat chosen from TebPlotter::FileFormat enum.

Examples:

integrator_system_classic_mpc.cpp.

Definition at line 158 of file teb_plotter.h.

References _file_export, _pdf_flag, EPS, JPEG, PDF, pipe, PRINT_DEBUG, PRINT_DEBUG_ONCE, SVG, and TIKZ.

16.40.4.15 void teb::TebPlotter::setOutputToWindow ( ) [inline]

Definition at line 226 of file teb_plotter.h.

References pipe.

Referenced by TebPlotter().

16.40.4.16 void teb::TebPlotter::spyMatrix (const Eigen::Ref< const Eigen::MatrixXd > & *matrix*)

This function visualizes the non-zeros of a given sparse matrix with a marker at the specific non-zero position.

Parameters

matrix	Matrix given as an Eigen::Matrix data type.

Definition at line 529 of file teb_plotter.cpp.

References pipe, and PRINT_INFO.

16.40.4.17 void teb::TebPlotter::switchWindow (int *window_id,* bool *reset* = `true`) [inline]

Parameters

window_id	Switch to gnuplot window with id `window_id`. If it does not exist, create new window.

reset	Reset window settings (recommand after switching from multiplots with extra sizes)

Examples:

linear_system_ode_ctrl_comparison.cpp.

Definition at line 133 of file teb_plotter.h.

References _file_export, and pipe.

16.40.5 Member Data Documentation

16.40.5.1 bool teb::TebPlotter::_file_export = false [protected]

Definition at line 251 of file teb_plotter.h.

Referenced by isExportToFileEnabled(), setOutputToFile(), and switchWindow().

16.40.5.2 bool teb::TebPlotter::_pdf_flag = false [protected]

Definition at line 250 of file teb_plotter.h.

Referenced by completePdfExport(), and setOutputToFile().

16.40.5.3 FILE∗ teb::TebPlotter::pipe = nullptr [protected]

Definition at line 253 of file teb_plotter.h.

Referenced by clearWindow(), closeWindow(), completePdfExport(), plot(), plot1DVector(), plotCustomKey(), plotTEB(), plotTwoCol(), setOutputToFile(), setOutputToWindow(), spyMatrix(), switchWindow(), TebPlotter(), and ∼TebPlotter().

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

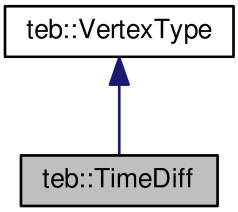
- teb_plotter.h
- teb_plotter.hpp
- teb_plotter.cpp

16.41 teb::TimeDiff Class Reference

Vertex that contains the time information of the TEB: ΔT.

`#include <teb_vertices.h>`

Inheritance diagram for teb::TimeDiff:



Public Types

- using VertexContainer = std::vector< VertexType * >

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).

Public Member Functions

- TimeDiff (bool fixed=false)

  Construct new TimeDiff and optinally set vertex fixed.
- TimeDiff (double dt, bool fixed=false)

  Construct new TimeDiff with specified initial value and optinally set vertex fixed.
- TimeDiff (double *dt, bool fixed=false)

  Construct new TimeDiff with specified initial value and optinally set vertex fixed.
- TimeDiff (const TimeDiff &obj)

  Copy constructor.
- ∼TimeDiff ()
- virtual bool isFixedAll () const

  return `true`, if ALL values of this vertex are fixed and therefore are not necessary for optimization.
- virtual void setFixedAll (bool fixed)

  set VertexType fixed.
- virtual bool isFixedAny () const

  return `true`, if ANY element/value/component of this vertex is fixed and therefore the vertex is relevant for optimization.
- virtual bool isFixedComp (int) const

  return `true`, if element/value/component with index `idx` of this vertex is fixed.
- double & dt ()

  return current ΔT.
- const double & dt () const

  return current ΔT (read-only).
- virtual void push ()

  This function should store all values into a internal backup stack.
- virtual void pop ()

  This function should restore the previously stored values of the backup stack and remove them from the stack.
- virtual void top ()

  This function should restore the previously stored values of the backup stack WITHOUT removing them from the stack.

- virtual void discardTop ()

  This function should delete the previously made backup from the stack without restoring it.
- virtual unsigned int stackSize () const

  This function should return the current size/number of backups of the backup stack.
- virtual int dimension () const

  Return number of elements/values/components stored in this vertex.
- virtual int dimensionFree () const

  Return only the dimension of free (unfixed) variables.
- virtual void plus (const VertexType ∗rhs)

  Define the increment for the vertex: x = x + rhs.
- virtual void plus (const double ∗rhs)

  Define the increment for the vertex: x = x + rhs (rhs[] with dimension()=p+q values)
- virtual void plusFree (const double ∗rhs)

  Define the increment for the vertex: x = x + rhs (But only add FREE variables, rhs[] with dimensionFree() values).
- virtual void setFree (const double ∗rhs)

  Overwrite all free variables with the values given by rhs (rhs[] with dimensionFree() values).
- virtual void getDataFree (double ∗target_vec) const

  Return a copy of the free (unfixed) values as a single array (length: dimensionFree()).
- virtual const double & getData (unsigned int) const

  Return pointer to data with the index `idx` *).*
- TimeDiff & operator= (const TimeDiff &rhs)
- void operator() (double &dt)
- void setOptVecIdx (int opt_vec_idx)

  Update the position of the first value of this vertex inside the optimization vector / Hessian.
- int getOptVecIdx () const

  Get the position of the first value of this Vertex inside the optimization vector / Hessian.

Static Public Attributes

- static const int Dimension = 1

  The dimension of the TimeDiff vertex is always 1.

Protected Attributes

- double _dt = 0.1
- bool _fixed = false
- std::stack< double > _backup
- int _opt_vec_idx = -1

  Start-index in optimization vector (idx in hessian (row/column) and jacobian (column))

Friends

- std::ostream & operator<< (std::ostream &os, const TimeDiff &dt)

  Print ΔT using std::ostream.

16.41.1 Detailed Description

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

VertexType, StateVertex, ControlVertex, BaseEdge

Definition at line 518 of file teb_vertices.h.

16.41.2 Member Typedef Documentation

16.41.2.1 using **teb::VertexType::VertexContainer** = std::vector<**VertexType**∗> [inherited]

Definition at line 33 of file graph.h.

16.41.3 Constructor & Destructor Documentation

16.41.3.1 teb::TimeDiff::TimeDiff (bool *fixed* = false) [inline]

Parameters

fixed	if `true` , the TimeDiff is fixed during optimization (no time-optimal control possible).

Definition at line 527 of file teb_vertices.h.

References _fixed.

16.41.3.2 teb::TimeDiff::TimeDiff (double *dt*, bool *fixed* = false) [inline]

Parameters

dt	initial ΔT
fixed	if `true` , the TimeDiff is fixed during optimization (no time-optimal control possible).

Definition at line 534 of file teb_vertices.h.

16.41.3.3 teb::TimeDiff::TimeDiff (double ∗ *dt*, bool *fixed* = false) [inline]

Parameters

dt	Pointer to initial ΔT (will be copied)
fixed	if `true` , the TimeDiff is fixed during optimization (no time-optimal control possible).

Definition at line 541 of file teb_vertices.h.

16.41.3.4 teb::TimeDiff::TimeDiff (const **TimeDiff** & *obj*) [inline]

Parameters

obj	TimeDiff object

Definition at line 547 of file teb_vertices.h.

16.41.3.5 teb::TimeDiff::∼TimeDiff ( ) [inline]

Definition at line 549 of file teb_vertices.h.

16.41.4 Member Function Documentation

16.41.4.1 virtual int teb::TimeDiff::dimension ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 574 of file teb_vertices.h.

16.41.4.2 virtual int teb::TimeDiff::dimensionFree ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 575 of file teb_vertices.h.

References _fixed.

16.41.4.3 virtual void teb::TimeDiff::discardTop ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 571 of file teb_vertices.h.

References _backup.

16.41.4.4 double& teb::TimeDiff::dt ( ) [inline]

Definition at line 556 of file teb_vertices.h.

References _dt.

Referenced by teb::EdgeMinimizeTime::computeValues(), teb::TebController< p, q >::getDt(), operator()(), operator=(), plus(), teb::TebController< p, q >::resetController(), and teb::TebController< p, q >::setupHorizon().

16.41.4.5 const double& teb::TimeDiff::dt ( ) const [inline]

Definition at line 557 of file teb_vertices.h.

References _dt.

16.41.4.6 virtual const double& teb::TimeDiff::getData (unsigned *idx*) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 609 of file teb_vertices.h.

References _dt.

16.41.4.7 virtual void teb::TimeDiff::getDataFree (double * *target_vec*) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 600 of file teb_vertices.h.

References _dt, and _fixed.

16.41.4.8 int teb::VertexType::getOptVecIdx ( ) const [inline], [inherited]

Returns

position of the first element of this vertex.

Definition at line 59 of file graph.h.

References teb::VertexType::_opt_vec_idx.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), and teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ().

16.41.4.9 virtual bool teb::TimeDiff::isFixedAll ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 551 of file teb_vertices.h.

References _fixed.

Referenced by operator=().

16.41.4.10 virtual bool teb::TimeDiff::isFixedAny ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 553 of file teb_vertices.h.

References _fixed.

16.41.4.11 virtual bool teb::TimeDiff::isFixedComp (int *idx*) const [inline], [virtual]

Parameters

idx	component index

Implements teb::VertexType.

Definition at line 554 of file teb_vertices.h.

References _fixed.

16.41.4.12 void teb::TimeDiff::operator() (double & *dt*) [inline]

Definition at line 625 of file teb_vertices.h.

References _dt, and dt().

16.41.4.13 TimeDiff& teb::TimeDiff::operator= (const TimeDiff & *rhs*) [inline]

Definition at line 615 of file teb_vertices.h.

References _dt, _fixed, dt(), and isFixedAll().

16.41.4.14 virtual void teb::TimeDiff::plus (const VertexType ∗ *rhs*) [inline], [virtual]

Implements teb::VertexType.

Definition at line 578 of file teb_vertices.h.

References _dt, and dt().

16.41.4.15 virtual void teb::TimeDiff::plus (const double ∗ *rhs*) [inline], [virtual]

Implements teb::VertexType.

Definition at line 584 of file teb_vertices.h.

References _dt.

16.41.4.16 virtual void teb::TimeDiff::plusFree (const double ∗ *rhs*) [inline], [virtual]

Implements teb::VertexType.

Definition at line 589 of file teb_vertices.h.

References _dt, and _fixed.

16.41.4.17 virtual void teb::TimeDiff::pop ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 561 of file teb_vertices.h.

References _backup, and top().

16.41.4.18 virtual void teb::TimeDiff::push ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 560 of file teb_vertices.h.

References _backup, and _dt.

16.41.4.19 virtual void teb::TimeDiff::setFixedAll (bool *fixed*) [inline], [virtual]

Parameters

fixed	if `true` TimeDiff is fixed.

Definition at line 552 of file teb_vertices.h.

References _fixed.

Referenced by teb::TebController< p, q >::setFixedDt().

16.41.4.20 virtual void teb::TimeDiff::setFree (const double ∗ *rhs*) [inline], [virtual]

Implements teb::VertexType.

Definition at line 594 of file teb_vertices.h.

References _dt, and _fixed.

16.41.4.21 void teb::VertexType::setOptVecIdx (int *opt_vec_idx*) [inline], [inherited]

Parameters

opt_vec_idx	index of the first value

Definition at line 53 of file graph.h.

References teb::VertexType::_opt_vec_idx.

16.41.4.22 virtual unsigned int teb::TimeDiff::stackSize ( ) const [inline], [virtual]

Implements teb::VertexType.

Definition at line 572 of file teb_vertices.h.

References _backup.

16.41.4.23 virtual void teb::TimeDiff::top ( ) [inline], [virtual]

Implements teb::VertexType.

Definition at line 566 of file teb_vertices.h.

References _backup, and _dt.

Referenced by pop().

16.41.5 Friends And Related Function Documentation

16.41.5.1 std::ostream& operator<< (std::ostream & *os*, const TimeDiff & *dt*) [friend]

Parameters

os	output stream object
dt	specific TimeDiff object

Returns

output stream object including a formatted string containing state and control vector.

Definition at line 637 of file teb_vertices.h.

16.41.6 Member Data Documentation

16.41.6.1 std::stack<double> teb::TimeDiff::_backup [protected]

Definition at line 646 of file teb_vertices.h.

Referenced by discardTop(), pop(), push(), stackSize(), and top().

16.41.6.2 double teb::TimeDiff::_dt = 0.1 [protected]

Definition at line 644 of file teb_vertices.h.

Referenced by dt(), getData(), getDataFree(), operator()(), operator=(), plus(), plusFree(), push(), setFree(), and top().

16.41.6.3 bool teb::TimeDiff::_fixed = false [protected]

Definition at line 645 of file teb_vertices.h.

Referenced by dimensionFree(), getDataFree(), isFixedAll(), isFixedAny(), isFixedComp(), operator=(), plusFree(), setFixedAll(), setFree(), and TimeDiff().

16.41.6.4 int teb::VertexType::_opt_vec_idx = -1 [protected], [inherited]

Definition at line 69 of file graph.h.

Referenced by teb::VertexType::getOptVecIdx(), and teb::VertexType::setOptVecIdx().

16.41.6.5 const int teb::TimeDiff::Dimension = 1 [static]

Definition at line 521 of file teb_vertices.h.

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

- teb_vertices.h

16.42 teb::SimResults::TimeSeries Struct Reference

Store measurements of dynamic systems (states and control inputs) w.r.t.

`#include <simulator.h>`

Public Attributes

- Eigen::MatrixXd states

    Contains state vectors [p x n].
- Eigen::MatrixXd controls

    Contains control input vectors [q x n].
- Eigen::VectorXd time

    Time vector [0 .. T] [n x 1].
- double dt

    Store time difference ΔT of the current TEB.

16.42.1 Detailed Description

time The time is discretized into *n* discrete time samples

Definition at line 37 of file simulator.h.

16.42.2 Member Data Documentation

16.42.2.1 Eigen::MatrixXd teb::SimResults::TimeSeries::controls

Definition at line 40 of file simulator.h.

Referenced by teb::SimResults::allocateMemory(), and teb::SimResults::conservativeResize().

16.42.2.2 double teb::SimResults::TimeSeries::dt

Definition at line 42 of file simulator.h.

16.42.2.3 Eigen::MatrixXd teb::SimResults::TimeSeries::states

Definition at line 39 of file simulator.h.

Referenced by teb::SimResults::allocateMemory(), and teb::SimResults::conservativeResize().

16.42.2.4 Eigen::VectorXd teb::SimResults::TimeSeries::time

Definition at line 41 of file simulator.h.

Referenced by teb::SimResults::allocateMemory(), and teb::SimResults::conservativeResize().

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

- simulator.h

16.43 teb::Config::Utilities Struct Reference

Configurations for helper miscellaneous and utilities.

```
#include <config.h>
```

16.43.1 Detailed Description

Definition at line 138 of file config.h.

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

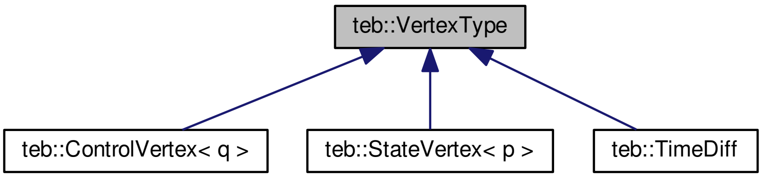
- config.h

16.44 teb::VertexType Class Reference

Generic interface class for vertices.

```
#include <graph.h>
```

Inheritance diagram for teb::VertexType:



Public Types

- using VertexContainer = std::vector< VertexType ∗ >

  Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).

Public Member Functions

- virtual bool isFixedAll () const =0

  return `true`, if ALL values of this vertex are fixed and therefore are not necessary for optimization.
- virtual bool isFixedAny () const =0

  return `true`, if ANY element/value/component of this vertex is fixed and therefore the vertex is relevant for optimization.
- virtual bool isFixedComp (int idx) const =0

  return `true`, if element/value/component with index `idx` of this vertex is fixed.
- virtual void push ()=0

  This function should store all values into a internal backup stack.
- virtual void pop ()=0

  This function should restore the previously stored values of the backup stack and remove them from the stack.
- virtual void top ()=0

  This function should restore the previously stored values of the backup stack WITHOUT removing them from the stack.
- virtual void discardTop ()=0

  This function should delete the previously made backup from the stack without restoring it.
- virtual unsigned int stackSize () const =0

  This function should return the current size/number of backups of the backup stack.
- virtual int dimension () const =0

  Return number of elements/values/components stored in this vertex.
- virtual int dimensionFree () const =0

  Return only the dimension of free (unfixed) variables.
- void setOptVecIdx (int opt_vec_idx)

  Update the position of the first value of this vertex inside the optimization vector / Hessian.
- int getOptVecIdx () const

  Get the position of the first value of this Vertex inside the optimization vector / Hessian.
- virtual void plus (const VertexType ∗rhs)=0

  Define the increment for the vertex: x = x + rhs.
- virtual void plus (const double ∗rhs)=0

  Define the increment for the vertex: x = x + rhs (rhs[] with dimension()=p+q values)

- virtual void plusFree (const double ∗rhs)=0

  Define the increment for the vertex: x = x + rhs (But only add FREE variables, rhs[] with dimensionFree() values).
- virtual void setFree (const double ∗rhs)=0

  Overwrite all free variables with the values given by rhs (rhs[] with dimensionFree() values).
- virtual void getDataFree (double ∗target_vec) const =0

  Return a copy of the free (unfixed) values as a single array (length: dimensionFree()).
- virtual const double & getData (unsigned int idx) const =0

  Return pointer to data with the index `idx`).

Protected Attributes

- int _opt_vec_idx = -1

  Start-index in optimization vector (idx in hessian (row/column) and jacobian (column))

16.44.1 Detailed Description

This abstract class defines the interface for dedicated vertices. The underlying TEB optimization problem is formulated as a hyper-graph in which state vectors (StateVertex), control input vectors (ControlVertex) and a TimeDiff are represented as vertices. That means, vertices have to be considered by the solver as changeable quantities.

Author

Christoph Rösmann (christoph.roesmann@tu-dortmund.de)

See Also

HyperGraph, EdgeType, StateVertex, ControlVertex, TimeDiff, BaseEdge

Definition at line 29 of file graph.h.

16.44.2 Member Typedef Documentation

16.44.2.1 using teb::VertexType::VertexContainer = std::vector<VertexType∗>

Definition at line 33 of file graph.h.

16.44.3 Member Function Documentation

16.44.3.1 virtual int teb::VertexType::dimension ( ) const [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::BaseEdge< D, FuncType, Vertices >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::BaseEdge< D, FuncType, Vertices >::computeJacobian(), teb::BaseEdge< D, FuncType >::computeJacobian(), and teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ().

16.44.3.2 virtual int teb::VertexType::dimensionFree ( ) const [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

16.44.3.3 virtual void teb::VertexType::discardTop ( ) [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

16.44.3.4 virtual const double& teb::VertexType::getData (unsigned int *idx*) const [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

16.44.3.5 virtual void teb::VertexType::getDataFree (double * *target_vec*) const [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

16.44.3.6 int teb::VertexType::getOptVecIdx ( ) const [inline]

Returns

position of the first element of this vertex.

Definition at line 59 of file graph.h.

References _opt_vec_idx.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), and teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ().

16.44.3.7 virtual bool teb::VertexType::isFixedAll ( ) const [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

Referenced by teb::BaseEdge< D, FuncType, Vertices >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::BaseEdge< D, FuncType, Vertices >::computeJacobian(), and teb::BaseEdge< D, FuncType >::computeJacobian().

16.44.3.8 virtual bool teb::VertexType::isFixedAny ( ) const [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient(), and teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically().

16.44.3.9 virtual bool teb::VertexType::isFixedComp (int *idx*) const [pure virtual]

Parameters

idx	component index

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

Referenced by teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian(), teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverNonlinear-

ProgramDense::buildObjectiveGradient(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), and teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ().

16.44.3.10 virtual void teb::VertexType::plus (const VertexType * *rhs*) [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

Referenced by teb::BaseEdge< D, FuncType, Vertices >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::BaseEdge< D, FuncType, Vertices >::computeJacobian(), and teb::BaseEdge< D, FuncType >::computeJacobian().

16.44.3.11 virtual void teb::VertexType::plus (const double * *rhs*) [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

16.44.3.12 virtual void teb::VertexType::plusFree (const double * *rhs*) [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

16.44.3.13 virtual void teb::VertexType::pop ( ) [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

Referenced by teb::BaseEdge< D, FuncType, Vertices >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::BaseEdge< D, FuncType, Vertices >::computeJacobian(), and teb::BaseEdge< D, FuncType >::computeJacobian().

16.44.3.14 virtual void teb::VertexType::push ( ) [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

Referenced by teb::BaseEdge< D, FuncType, Vertices >::computeHessian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::BaseEdge< D, FuncType, Vertices >::computeJacobian(), and teb::BaseEdge< D, FuncType >::computeJacobian().

16.44.3.15 virtual void teb::VertexType::setFree (const double * *rhs*) [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

16.44.3.16 void teb::VertexType::setOptVecIdx (int *opt_vec_idx*) [inline]

Parameters

opt_vec_idx	index of the first value

Definition at line 53 of file graph.h.

References _opt_vec_idx.

16.44.3.17 virtual unsigned int teb::VertexType::stackSize ( ) const [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

16.44.3.18 virtual void teb::VertexType::top ( ) [pure virtual]

Implemented in teb::TimeDiff, teb::ControlVertex< q >, and teb::StateVertex< p >.

16.44.4 Member Data Documentation

16.44.4.1 int teb::VertexType::_opt_vec_idx = -1 [protected]

Definition at line 69 of file graph.h.

Referenced by getOptVecIdx(), and setOptVecIdx().

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

- graph.h

Chapter 17

File Documentation

17.1 base_controller.h File Reference

```
#include <teb_package/base/config.h>
#include <memory>
#include <cmath>
#include <deque>
```

Classes

- class teb::BaseController

  Base controller class (General Controller API)

Namespaces

- teb

  General project namespace for all library functions and objects.

17.2 base_edge.h File Reference

```
#include <teb_package/base/graph.h>
#include <teb_package/base/teb_vertices.h>
#include <teb_package/utilities/misc.h>
#include <array>
#include <teb_package/base/base_edge.hpp>
```

Classes

- class teb::BaseEdge< D, FuncType, Vertices >

  Templated base edge class that stores an arbitary number of values.

- class teb::BaseEdge< D, FuncType >

  Templated base edge class that stores an arbitary number of values (Partial template specialization that allows changing vertices dimensions at runtime).

Namespaces

- teb

    General project namespace for all library functions and objects.

17.3 base_edge.hpp File Reference

```
#include <teb_package/base/base_edge.h>
```

Namespaces

- teb

    General project namespace for all library functions and objects.

17.4 base_edge_dynamics.h File Reference

```
#include <teb_package/base/base_edge.h>
#include <teb_package/base/system_dynamics.h>
```

Classes

- class teb::EdgeSystemDynamics< p, q, central >

    Equality constraint edge for satisfying continious system dynamics specified by an SystemDynamics object.

Namespaces

- teb

    General project namespace for all library functions and objects.

17.5 base_solver.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/config.h>
#include <teb_package/base/graph.h>
#include <memory>
#include <cmath>
#include <deque>
#include <Eigen/Core>
#include <Eigen/StdVector>
```

Classes

- class teb::BaseSolver

    Base class for solver implementations.

Namespaces

- teb

  General project namespace for all library functions and objects.

17.6 base_solver_least_squares.cpp File Reference

```
#include <teb_package/base/base_solver_least_squares.h>
```

Namespaces

- teb

  General project namespace for all library functions and objects.

17.7 base_solver_least_squares.h File Reference

```
#include <teb_package/base/base_solver.h>
#include <Eigen/Dense>
```

Classes

- class teb::BaseSolverLeastSquares

  Extended base solver class for least squares optimizations.

Namespaces

- teb

  General project namespace for all library functions and objects.

17.8 base_solver_nonlinear_program_dense.cpp File Reference

```
#include <teb_package/solver/base_solver_nonlinear_program_dense.h>
```

Namespaces

- teb

  General project namespace for all library functions and objects.

17.9 base_solver_nonlinear_program_dense.h File Reference

```
#include <teb_package/base/base_solver.h>
#include <Eigen/Dense>
```

Classes

- class teb::BaseSolverNonlinearProgramDense

  Extended base solver class for nonlinear programs (dense version).

Namespaces

- teb

  General project namespace for all library functions and objects.

17.10 bound_constraints.h File Reference

```
#include <teb_package/base/base_edge.h>
#include <type_traits>
#include <tuple>
```

Classes

- struct teb::CustomBoundData

  Datastruct for custom bound data that can be queried from the BoundConstraint class.
- class teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >

  This class captures general bound constraints on optimization variables / vertices.

Namespaces

- teb

  General project namespace for all library functions and objects.

Enumerations

- enum teb::BOUND_TYPE { teb::BOUND_TYPE::LOWER, teb::BOUND_TYPE::UPPER, teb::BOUND_TYPE::LOWERUPPER }

  Enumerator class that stores all types of bounds for consideration in BoundConstraint class.
- enum teb::BOUND_VARS { teb::BOUND_VARS::ALL, teb::BOUND_VARS::SINGLE }

  Enumerator class that stores all types of variables that can be bounded using BoundConstraint class.

17.11 common_teb_edges.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/bound_constraints.h>
#include <teb_package/base/base_edge_dynamics.h>
#include <teb_package/base/system_dynamics.h>
```

Classes

- class teb::EdgeMinimizeTime

  Objective edge: minimize ΔT.
- class teb::EdgeQuadraticForm< p, q >

  Objective edge: quadratic form for states and control input.

Namespaces

- teb

  General project namespace for all library functions and objects.

Typedefs

- using teb::EdgePositiveTime = BoundConstraint< BOUND_TYPE::LOWER, TimeDiff, BOUND_VARS::SINGLE, 0 >

  Bound constraint edge for the time difference: $\Delta T \geq \varepsilon$.
- template<int q>
  using teb::EdgeControlBounds = BoundConstraint< BOUND_TYPE::LOWERUPPER, ControlVertex< q >, BOUND_VARS::ALL >

  Bound constraint edge for control inputs: $\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}$.
- template<int p>
  using teb::EdgeStateBounds = BoundConstraint< BOUND_TYPE::LOWERUPPER, StateVertex< p >, BOUND_VARS::ALL >

  Bound constraint edge for states: $\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}$.

17.12 config.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <cstdint>
```

Classes

- class teb::Config

  Configurations for Controller and Solver classes.
- struct teb::Config::Teb

  Configurations related to the TEB algorithm.
- struct teb::Config::Optim

  Configurations related to the trajectory optimization.
- struct teb::Config::Optim::Solver

  Configurations related to solvers.
- struct teb::Config::Optim::Solver::Lsq

  Configurations especially for least-squares-solvers.
- struct teb::Config::Optim::Solver::NonlinearProgram

  Settings for constrained optimiziation solver (nonlinear program solver)
- struct teb::Config::Optim::Solver::NonlinearProgram::Hessian

  Configurations for the hessian of the lagrangian calculation.
- struct teb::Config::Optim::Solver::NonlinearProgram::LineSearch

  Settings for the line-search algorithm (force merit function descent)

- struct teb::Config::Optim::Solver::Nlopt

  Configurations especially for the nlopt solver wrapper.
- struct teb::Config::Utilities

  Configurations for helper miscellaneous and utilities.

Namespaces

- teb

  General project namespace for all library functions and objects.

Enumerations

- enum teb::FiniteDifferences { teb::FiniteDifferences::FORWARD, teb::FiniteDifferences::CENTRAL }

  Enumeration for different finite difference types used especially for dynamic system discretization.
- enum teb::HessianMethod {
  teb::HessianMethod::NUMERIC, teb::HessianMethod::BLOCK_BFGS, teb::HessianMethod::FULL_BFGS,
  teb::HessianMethod::FULL_BFGS_WITH_STRUCTURE_FILTER,
  teb::HessianMethod::ZERO_HESSIAN }

  Enumeration for different hessian calculation methods (not necessary for least-squares solver)
- enum teb::HessianInit { teb::HessianInit::ZERO, teb::HessianInit::IDENTITY, teb::HessianInit::NUMERIC }

  Enumeration for the hessian initialization (relevant for BFGS hessian approximations)
- enum teb::NloptAlgorithms { teb::NloptAlgorithms::SLSQP }

  Enumeration for different solver algorithms supported by the Nlopt library (only those that are working here)

17.13 download.md File Reference

17.14 graph.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/workspaces.h>
#include <Eigen/Core>
#include <stack>
#include <vector>
#include <memory>
```

Classes

- class teb::VertexType

  Generic interface class for vertices.
- class teb::EdgeType

  Generic interface class for edges.
- class teb::HyperGraph

  Graph object that stores pointers to active vertices and edges.

Namespaces

- teb

  General project namespace for all library functions and objects.

Enumerations

- enum teb::FUNCT_TYPE {
teb::FUNCT_TYPE::NONLINEAR, teb::FUNCT_TYPE::NONLINEAR_SQUARED, teb::FUNCT_TYPE::NONLINEAR_ONCE_DIFF, teb::FUNCT_TYPE::QUADRATIC,
teb::FUNCT_TYPE::LINEAR_SQUARED, teb::FUNCT_TYPE::LINEAR }

  Enum for different function types (see EdgeType class)

17.15 groups.dox File Reference

17.16 install.md File Reference

17.17 integrators.h File Reference

```
#include <teb_package/base/system_dynamics.h>
#include <Eigen/Core>
#include <queue>
```

Classes

- class teb::NumericalIntegrator< p, q >

  Interface for numerical integration methods used in simulation.
- class teb::ExplicitEuler< p, q >

  Simple explicit euler method for integration.
- class teb::RungeKuttaClassic< p, q >

  Classic Runge Kutta method (fourth order)
- class teb::RungeKutta5thOrder< p, q >

  Runge Kutta method (fifth order, slighly modified)

Namespaces

- teb

  General project namespace for all library functions and objects.

Enumerations

- enum teb::NumericalIntegrators { teb::NumericalIntegrators::EXPLICIT_EULER, teb::NumericalIntegrators::RUNGE_KUTTA_CLASSIC, teb::NumericalIntegrators::RUNGE_KUTTA_5TH }

  Enumeration for different numerical integration methods.

17.18 mainpage.dox File Reference

17.19 matlab_class_handle.hpp File Reference

17.20 matlab_interface.md File Reference

17.21 measure_cpu_time.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <chrono>
```

Macros

- #define START_TIMER auto start = std::chrono::high_resolution_clock::now();
- #define STOP_TIMER(name)

17.21.1 Macro Definition Documentation

17.21.1.1 #define START_TIMER auto start = std::chrono::high_resolution_clock::now();

Examples:

integrator_system1.cpp, linear_system_ode.cpp, linear_system_state_space.cpp, mobile_robot_teb.cpp, and van_der_pol_system.cpp.

Definition at line 12 of file measure_cpu_time.h.

17.21.1.2 #define STOP_TIMER(*name*)

Value:

```
PRINT_INFO("RUNTIME of " << name << ": " << \
    std::chrono::duration_cast<std::chrono::milliseconds>( \
        std::chrono::high_resolution_clock::now()-start \
    ).count() << " ms " << "(Compiled in Debug mode)");
```

Examples:

integrator_system1.cpp, linear_system_ode.cpp, linear_system_state_space.cpp, mobile_robot_teb.cpp, and van_der_pol_system.cpp.

Definition at line 19 of file measure_cpu_time.h.

17.22 misc.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <math.h>
#include <functional>
#include <tuple>
```

Functions

- double norm_angle (double angle)

  Normalize angle to interval [-pi, pi)
- void norm_angle_vec (Eigen::Ref< Eigen::MatrixXd > angles)

  Normalize vector or matrix of angles to interval [-pi, pi)

17.22.1 Function Documentation

17.22.1.1 double norm_angle (double *angle*) [inline]

Parameters

angle	angle in radiant

Returns

normalized angle to [-pi, pi) in rad

Examples:

mobile_robot_teb.cpp.

Definition at line 15 of file misc.h.

References PI.

Referenced by norm_angle_vec().

17.22.1.2 void norm_angle_vec (Eigen::Ref< Eigen::MatrixXd > *angles*) [inline]

Parameters

angles	angle vector/matrix in radiant [all values will be replaced by its normalized version]

Examples:

mobile_robot_teb.cpp.

Definition at line 35 of file misc.h.

References norm_angle().

17.23 simulator.h File Reference

```
#include <teb_package/base/base_controller.h>
#include <teb_package/base/system_dynamics.h>
#include <teb_package/simulation/integrators.h>
#include <teb_package/visualization/teb_plotter.h>
#include <functional>
#include <teb_package/simulation/simulator.hpp>
```

Classes

- class teb::SimResults

  Simulation results are stored and combined in this container class.
- struct teb::SimResults::TimeSeries

  Store measurements of dynamic systems (states and control inputs) w.r.t.
- class teb::Simulator< p, q >

  Simulator for dynamic systems controlled by a TebController.

Namespaces

- teb

  General project namespace for all library functions and objects.

17.24 simulator.hpp File Reference

```
#include <teb_package/simulation/simulator.h>
```

Namespaces

- teb

  General project namespace for all library functions and objects.

17.25 solver_levenbergmarquardt_eigen_dense.cpp File Reference

```
#include <teb_package/solver/solver_levenbergmarquardt_eigen_dense.h>
#include <iostream>
```

Namespaces

- teb

  General project namespace for all library functions and objects.

17.26 solver_levenbergmarquardt_eigen_dense.h File Reference

```
#include <teb_package/base/base_solver_least_squares.h>
```

Classes

- class teb::SolverLevenbergMarquardtEigenDense

  Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Dense matrices version).

Namespaces

- teb

  General project namespace for all library functions and objects.

17.27 solver_levenbergmarquardt_eigen_sparse.cpp File Reference

```
#include <teb_package/solver/solver_levenbergmarquardt_eigen_sparse.h>
#include <iostream>
```

Namespaces

- teb

  General project namespace for all library functions and objects.

17.28 solver_levenbergmarquardt_eigen_sparse.h File Reference

```
#include <teb_package/base/base_solver_least_squares.h>
#include <Eigen/Sparse>
```

Classes

- class teb::SolverLevenbergMarquardtEigenSparse

  Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Sparse matrices version).

Namespaces

- teb

  General project namespace for all library functions and objects.

17.29 solver_nlopt_package.cpp File Reference

17.30 solver_nlopt_package.h File Reference

```
#include <teb_package/base/base_solver.h>
```

Classes

- class teb::SolverNloptPackage

  This class wraps the optimization problem to allow solving by NLOPT.

Namespaces

- teb

  General project namespace for all library functions and objects.

17.31 solver_overview.md File Reference

17.32 solver_sqp_dense.h File Reference

```
#include <teb_package/solver/base_solver_nonlinear_program_dense.h>
#include <qpOASES.hpp>
```

Classes

- class teb::SolverSQPDense

  Dense sequential quadratic programming (SQP) solver for nonlinear programs.

Namespaces

- teb

  General project namespace for all library functions and objects.

Macros

- #define __SUPPRESSANYOUTPUT__

17.32.1 Macro Definition Documentation

17.32.1.1 #define __SUPPRESSANYOUTPUT__

Definition at line 6 of file solver_sqp_dense.h.

17.33 solver_sqp_dense_backup.h File Reference

```
#include <teb_package/solver/base_solver_nonlinear_program_dense.h>
#include <qpOASES.hpp>
```

Classes

- class teb::SolverSQPDense

  Dense sequential quadratic programming (SQP) solver for nonlinear programs.

Namespaces

- teb

  General project namespace for all library functions and objects.

17.34 system_dynamics.h File Reference

```
#include <Eigen/Core>
#include <Eigen/Dense>
```

Classes

- class teb::SystemDynamics< p, q >

  Helper class for modeling nonlinear dynamic systems.

Namespaces

- teb

  General project namespace for all library functions and objects.

17.35 teb_controller.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/base_controller.h>
#include <teb_package/base/teb_vertices.h>
#include <teb_package/base/common_teb_edges.h>
#include <teb_package/base/base_solver.h>
#include <Eigen/Core>
#include <Eigen/StdDeque>
#include "algorithm"
#include <teb_package/base/teb_controller.hpp>
```

Classes

- class teb::TebController< p, q >

    Main Timed-Elastic-Band controller class.

Namespaces

- teb

    General project namespace for all library functions and objects.

17.36 teb_controller.hpp File Reference

```
#include <teb_package/base/teb_controller.h>
```

Namespaces

- teb

    General project namespace for all library functions and objects.

17.37 teb_plotter.cpp File Reference

```
#include <teb_package/visualization/teb_plotter.h>
```

Namespaces

- teb

    General project namespace for all library functions and objects.

17.38 teb_plotter.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/teb_controller.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <teb_package/visualization/teb_plotter.hpp>
```

Classes

- struct teb::PlotOptions

    Customize plots and figures.
- class teb::TebPlotter

    Provides useful visualization and plotting functions.

Namespaces

- teb

    General project namespace for all library functions and objects.

17.39 teb_plotter.hpp File Reference

```
#include <teb_package/visualization/teb_plotter.h>
```

Namespaces

- teb

    General project namespace for all library functions and objects.

17.40 teb_vertices.h File Reference

```
#include <teb_package/base/graph.h>
```

Classes

- class teb::StateVertex< p >

    Vertex that contains the StateVector.
- class teb::ControlVertex< q >

    Vertex that contains the ControlVector.
- class teb::TimeDiff

    Vertex that contains the time information of the TEB: ΔT.

Namespaces

- teb

  General project namespace for all library functions and objects.

17.41 typedefs.h File Reference

```
#include <limits.h>
#include <iostream>
#include <vector>
#include <Eigen/Core>
#include <Eigen/StdVector>
#include <stack>
#include <deque>
```

Namespaces

- teb

  General project namespace for all library functions and objects.

Macros

- #define PI 3.14159265358979323846264

  Define π.
- #define INF HUGE_VAL

  Define Inifity ∞.
- #define PRINT_DEBUG(msg) std::cout << "Debug: " << msg << std::endl;

  Print `msg-stream` only if project is compiled in Debug-mode.
- #define PRINT_DEBUG_ONCE(msg) { static const auto debugOnce = [&] { std::cout << "Debug: " << msg << std::endl; return true;}(); (void)debugOnce; }

  Print `msg-stream` only once and only if project is compiled in Debug-mode.
- #define PRINT_DEBUG_COND(cond, msg) if (cond) std::cout << "Debug: " << msg << std::endl;

  Print `msg-stream` only if `cond` == `true` and only if project is compiled in Debug-mode.
- #define PRINT_DEBUG_COND_ONCE(cond, msg) { static const auto debugOnce = [&] { if (cond) std::cout << "Debug: " << msg << std::endl; return true;}(); (void)debugOnce; }

  Print `msg-stream` only if `cond` == `true`, only once and only if project is compiled in Debug-mode.
- #define PRINT_INFO(msg) std::cout << "Info: " << msg << std::endl;

  Print `msg-stream`.
- #define PRINT_INFO_ONCE(msg) { static const auto infoOnce = [&] { std::cout << "Info: " << msg << std::endl; return true;}(); (void)infoOnce; }

  Print `msg-stream` only once.
- #define PRINT_INFO_COND(cond, msg) if (cond) std::cout << "Info: " << msg << std::endl;

  Print `msg-stream` only if `cond` == `true`.
- #define PRINT_INFO_COND_ONCE(cond, msg) { static const auto infoOnce = [&] { if (cond) std::cout << "Info: " << msg << std::endl; return true;}(); (void)infoOnce; }

  Print `msg-stream` only if `cond` == `true`, only once.
- #define PRINT_ERROR(msg) std::cerr << "Error: " << msg << std::endl;

  Print `msg-stream` as error msg.
- #define INPUT_STREAM(variable, default_val) std::cin >> variable;

Typedefs

- template< typename T >
  using teb::EigenScalar = Eigen::Matrix< T, 1, 1 >

  Define 1x1 Eigen::Matrix.
- using teb::EigenScalarD = EigenScalar< double >

  Define 1x1 Eigen::Matrix of type double.
- template< typename T >
  using teb::BackupStackType = std::stack< T, std::deque< T > >

  Backup stack type.

17.41.1 Macro Definition Documentation

17.41.1.1 #define INF HUGE_VAL

Definition at line 39 of file typedefs.h.

Referenced by teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData(), and teb::-SolverSQPDense::initSolverWorkspace().

17.41.1.2 #define INPUT_STREAM(*variable, default_val*) std::cin >> variable;

Definition at line 105 of file typedefs.h.

Referenced by teb::Simulator< p, q >::simClosedLoop().

17.41.1.3 #define PI 3.14159265358979323846264

Definition at line 36 of file typedefs.h.

Referenced by norm_angle().

17.41.1.4 #define PRINT_DEBUG(*msg*) std::cout << "Debug: " << msg << std::endl;

Definition at line 63 of file typedefs.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::TebPlotter-::setOutputToFile(), and teb::SolverSQPDense::solveImpl().

17.41.1.5 #define PRINT_DEBUG_COND(*cond, msg*) if (cond) std::cout << "Debug: " << msg << std::endl;

Definition at line 69 of file typedefs.h.

17.41.1.6 #define PRINT_DEBUG_COND_ONCE(*cond, msg*) { static const auto debugOnce = [&] { if (cond) std::cout << "Debug: " << msg << std::endl; return true;}(); (void)debugOnce; }

Definition at line 72 of file typedefs.h.

Referenced by teb::TebController< p, q >::initOptimization(), teb::BaseSolverLeastSquares::initWorkspaces(), and teb::BaseSolverNonlinearProgramDense::initWorkspaces().

17.41.1.7 #define PRINT_DEBUG_ONCE(*msg*) { static const auto debugOnce = [&] { std::cout << "Debug: " << msg << std::endl; return true;}(); (void)debugOnce; }

Definition at line 66 of file typedefs.h.

Referenced by teb::TebPlotter::setOutputToFile(), and teb::SolverSQPDense::solveImpl().

17.41.1.8 #define PRINT_ERROR(*msg*) std::cerr << "Error: " << msg << std::endl;

Definition at line 103 of file typedefs.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), and teb::BaseSolverNonlinearProgramDense::initHessianBFGS().

17.41.1.9 #define PRINT_INFO(*msg*) std::cout << "Info: " << msg << std::endl;

Examples:

integrator_system1.cpp, integrator_system_classic_mpc.cpp, linear_system_ode.cpp, linear_system_state_space.cpp, and van_der_pol_system.cpp.

Definition at line 91 of file typedefs.h.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), teb::TebPlotter::plot(), teb::TebPlotter::plot1DVector(), teb::Simulator< p, q >::plotResults(), teb::TebPlotter::plotTEB(), teb::TebPlotter::plotTwoCol(), teb::Simulator< p, q >::simClosedLoop(), teb::TebPlotter::spyMatrix(), and teb::TebPlotter::TebPlotter().

17.41.1.10 #define PRINT_INFO_COND(*cond,* *msg*) if (cond) std::cout << "Info: " << msg << std::endl;

Definition at line 97 of file typedefs.h.

17.41.1.11 #define PRINT_INFO_COND_ONCE(*cond,* *msg*) { static const auto infoOnce = [&] { if (cond) std::cout << "Info: " << msg << std::endl; return true;}(); (void)infoOnce; }

Definition at line 100 of file typedefs.h.

17.41.1.12 #define PRINT_INFO_ONCE(*msg*) { static const auto infoOnce = [&] { std::cout << "Info: " << msg << std::endl; return true;}(); (void)infoOnce; }

Examples:

mobile_robot_teb.cpp.

Definition at line 94 of file typedefs.h.

Referenced by teb::Simulator< p, q >::simClosedLoop().

17.42 utilities.h File Reference

```
#include "misc.h"
#include "measure_cpu_time.h"
```

17.43 workspaces.h File Reference

```
#include <Eigen/Core>
#include <Eigen/StdVector>
#include <vector>
```

Classes

- class teb::JacobianWorkspace

  Memory management and accessor functions for the Block-Jacobians (for each edge).
- class teb::HessianWorkspace

  Memory management and accessor functions for the Block-Hessian (for each edge).

Namespaces

- teb

  General project namespace for all library functions and objects.

Chapter 18

Example Documentation

18.1 integrator_system1.cpp

The following example shows the TEB controller applied to an integrator system with *p* integrator states:

$$Tx^{(p)} = u$$

The control input u should be bounded to the interval [-1, 1].

Considering the particular case *p* = 4. The state vector is defined by $\mathbf{x} = [x, \dot{x}, \ddot{x}, \dddot{x}]^T$. Transforming ODE (1) into a state space corresponding to $\mathbf{x}$ leads to:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dddot{x} \\ \ddddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1/T \end{bmatrix} u$$

The objective of the control task is to transite the system from the start state $\mathbf{x}_s = [0,0,0,0]^T$ to the final state $\mathbf{x}_f = [1,0,0,0]^T$.

The system dynamics are specified using the IntegratorSystem class. See the following header file how to add the state space model mentioned above for an integrator with *p* states.

File **integrator_system.h:**

```
#ifndef __teb_package__examples_integrator_system__
#define __teb_package__examples_integrator_system__


#include <teb_package.h>


template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
  using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
  using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector
      ;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      StateVector state_equations;

      const int no_int = p-1;
      if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

      // add last equation containing control u
      state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
      return state_equations;
  }
```

```
    void setTimeConstant(double time_constant) {_time_constant = time_constant;};

protected:

    double _time_constant = 1;
};


#endif /* defined(__teb_package__examples_integrator_system__) */
```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the IntegratorSystem object specified in integrator_system.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows:

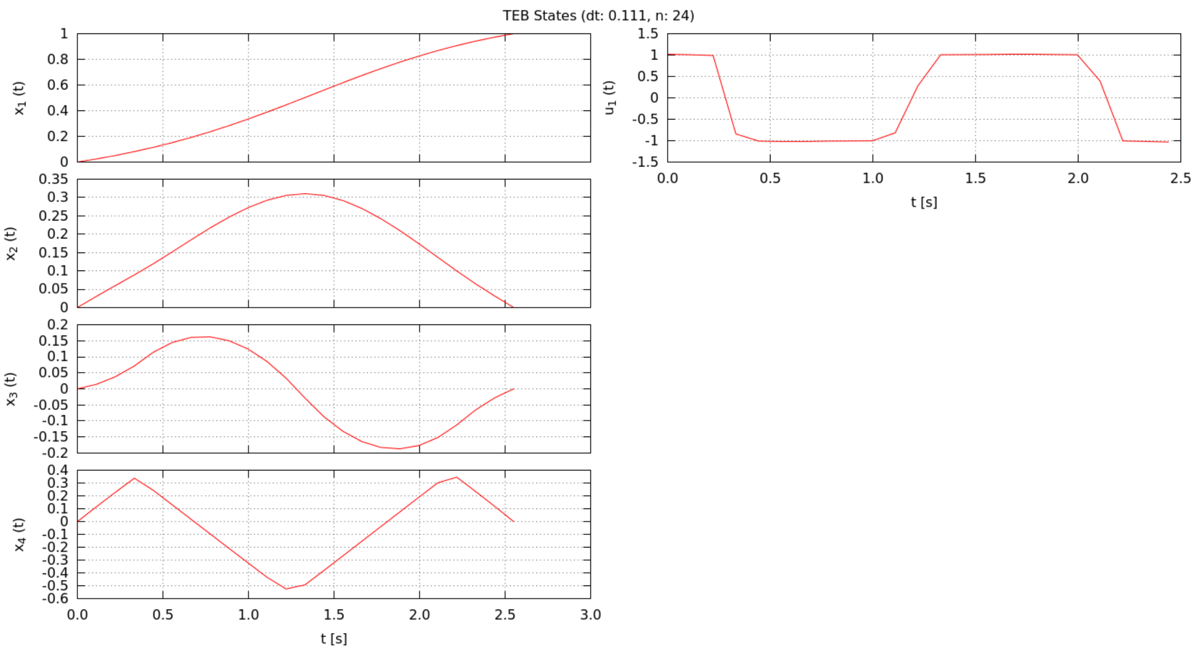


Figure 18.1: Result of integrator_system1.cpp

File **integrator_system1.cpp**

```
#include "integrator_system.h"

int main()
{
    const int p = 4;
    double x0[] = {0,0,0,0};
    double xf[] = {1,0,0,0};
    double u = 0;

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;


    // test SQP solver
    teb::Config cfg;
    //cfg.teb.teb_iter = 1;
    //cfg.optim.solver.nonlin_prog.hessian.hessian_method = teb::HessianMethod::FULL_BFGS;
    //cfg.optim.solver.nonlin_prog.hessian.hessian_init = teb::HessianInit::IDENTITY;
    //cfg.optim.solver.nonlin_prog.hessian.hessian_init_identity_scale = 0.5;
    //teb::SolverSQPDense solver;


    teb::TebController<p,1> teb(&cfg, &solver);
```

```
    // System specific settings
    IntegratorSystem<p> system;
    system.setTimeConstant(1.0);

    teb.setSystemDynamics(&system);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);

    // Perform open-loop planning
    START_TIMER;
    teb.step(x0, xf, &u);
    STOP_TIMER("TEB optimization loop");

    // Print determined control for the current sampling interval
    PRINT_INFO("Control u: " << u);

    // Plot states and control input
    teb::TebPlotter plotter;
    // optional: plotter.setOutputToFile("teb_opt_int",teb::TebPlotter::FileFormat::PNG);
    plotter.plotTEB(teb);


    return 0;
}
```

18.2 integrator_system2.cpp

The following example shows the TEB controller applied to an integrator system with *p* integrator states:

$$Tx^{(p)} = u$$

The control input u should be bounded to the interval [-1, 1].

Considering the particular case *p* = 2. The state vector is defined by $\mathbf{x} = [x, \dot{x}]^T$. Transforming ODE (1) into a state space corresponding to $\mathbf{x}$ leads to:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1/T \end{bmatrix} u$$

The objective of the control task is to transite the system from the start state $\mathbf{x}_s = [0,0]^T$ to the final state $\mathbf{x}_f = [1,0]^T$.

The system dynamics are specified using the IntegratorSystem class. See the following header file how to add the state space model mentioned above for an integrator with *p* states.

File **integrator_system.h:**

```
#ifndef __teb_package__examples_integrator_system__
#define __teb_package__examples_integrator_system__


#include <teb_package.h>


template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
  using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
  using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector
      ;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      StateVector state_equations;

      const int no_int = p-1;
      if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

      // add last equation containing control u
      state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
      return state_equations;
  }


    void setTimeConstant(double time_constant) {_time_constant = time_constant;};
```

```
protected:

    double _time_constant = 1;
};


#endif /* defined(__teb_package__examples_integrator_system__) */
```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the IntegratorSystem object specified in integrator_system.h.

Additionally, a Simulator object is initialized in order to perform closed-loop and open-loop simulations.

The example program compares the result for the open-loop control and the simulated closed-loop control. In addition the first optimziation result (red) is shown. If the system dynamic equation (equality constraint) is fully satisfied, red and green should be similar. They can only be identical, if the integrator method NumericalIntegrators-:ForwardEuler is used in simulation, since the TEB relies on the very same integration model.

The output of the program is as follows:

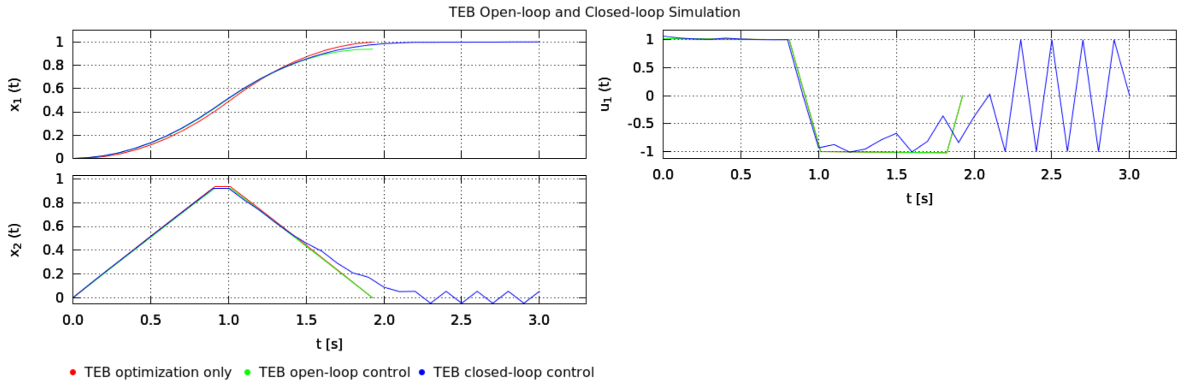


Figure 18.2: Result of integrator_system2.cpp

File **integrator_system2.cpp**

```
#include "integrator_system.h"


int main()
{

    const int p = 2; // number of states
    double x0[] = {0,0}; // start state
    double xf[] = {1,0}; // goal state

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&solver);

    // System specific settings
    IntegratorSystem<p> system;
    teb.setSystemDynamics(&system);

    // Optional: smooth control input trajectory
    // with small weighted quadratic cost
    //teb.activateObjectiveQuadraticForm(0,0.1,0);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);

    // Create simulator
    teb::TebPlotter plotter;
    teb::Simulator<p,1> sim(&teb,system,&plotter);
    // Set simulation sample time
    sim.setSampleTime(0.1);
    // Simulate 3 seconds
    sim.simOpenAndClosedLoop(x0,xf,3);
```

```
    return 0;
}
```

18.3 integrator_system_classic_mpc.cpp

The following example shows a classic MPC controller (with a quadratic cost functional and a receding horizon) applied to an integrator system with p integrator states:

$$T x^{(p)} = u$$

The control input u should be bounded to the interval [-1, 1].

Considering the particular case p = 3. The state vector is defined by $\mathbf{x} = [x, \dot{x}, \ddot{x}]^T$. Transforming ODE (1) into a state space corresponding to $\mathbf{x}$ leads to:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1/T \end{bmatrix} u$$

The objective of the control task is to transite the system from the start state $\mathbf{x}_s = [0, 0]^T$ to the final state $\mathbf{x}_f = [1, 0]^T$ while minimizing the quadratic control effort and control energy:

$$J = (\mathbf{x}_n - \mathbf{x}_f)^T \mathbf{Q}_f (\mathbf{x}_n - \mathbf{x}_f) + \sum_{k=0}^{n-1} [(\mathbf{x}_k - \mathbf{x}_f)^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_f) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k]$$

The receding/moving horizon is set to $n = 10$. The system dynamics are descretized using finite differences and with a uniform and fixed resolution ($\Delta T = 0.1$s).

The continuous system dynamics are specified using the IntegratorSystem class. See the following header file how to add the state space model mentioned above for an integrator with p states.

File **integrator_system.h:**

```
#ifndef __teb_package__examples_integrator_system__
#define __teb_package__examples_integrator_system__


#include <teb_package.h>


template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
  using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
  using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector
      ;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      StateVector state_equations;

      const int no_int = p-1;
      if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

      // add last equation containing control u
      state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
      return state_equations;
  }


    void setTimeConstant(double time_constant) {_time_constant = time_constant;};

protected:

    double _time_constant = 1;
};


#endif /* defined(__teb_package__examples_integrator_system__) */
```

The following cpp-file shows how to create the TebController object with common MPC characteristics, how to add control bounds and how to add the IntegratorSystem object specified in integrator_system.h.

Additionally, a Simulator object is initialized in order to perform closed-loop and open-loop simulations.

The example program compares the result for the open-loop control and the simulated closed-loop control. In addition the first optimziation result (red) is shown (the first horizon!).

The output of the program is as follows:

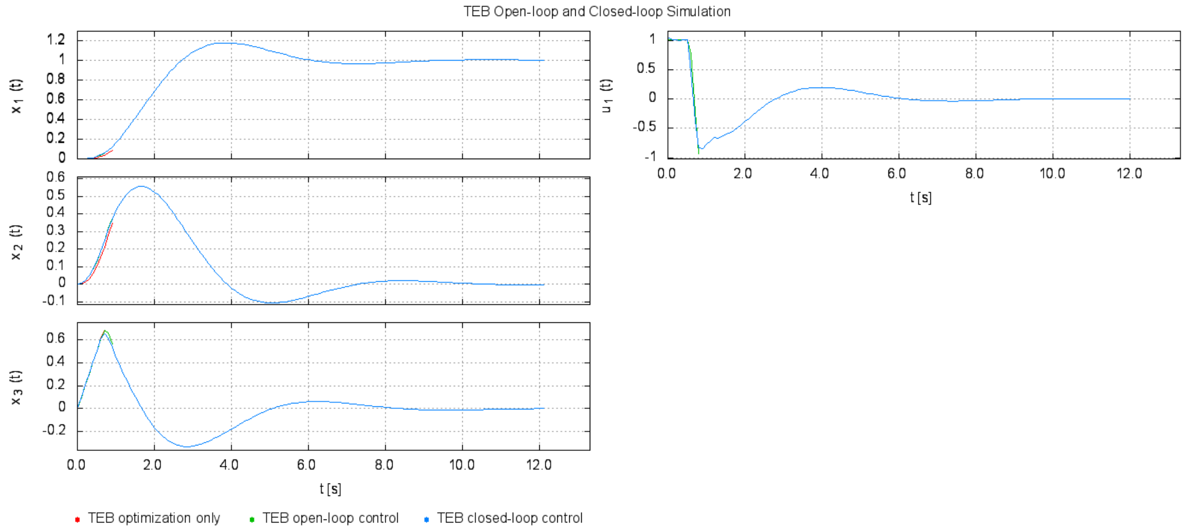


Figure 18.3: Result of integrator_system_classic_mpc.cpp

File **integrator_system_classic_mpc.cpp**

```
#include "integrator_system.h"

int main()
{
    const int p = 3;
    double x0[] = {0,0,0};
    double xf[] = {1,0,0};
    //double u; //! store control

    // Setup soft constraint weights
    teb::Config cfg;
    cfg.optim.solver.lsq.weight_equalities = 2;
    cfg.optim.solver.lsq.weight_inequalities = 2;
    cfg.optim.solver.lsq.weight_adaptation_factor = 5;


    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&cfg, &solver);

    // Setup receding horizon T = n * dt = 10 * 0.1
    // Common setting: unfixed goal and fixed resolution
    teb.setupHorizon(false, true, 10, 0.1);

    // Setup optimization problem
    teb.activateObjectiveQuadraticForm(0.5,0.1,50); // parameters: Q, R, Qf
    teb.activateControlBounds(0, -1.0, 1.0);

    IntegratorSystem<p> system;
    system.setTimeConstant(1.0);

    teb.setSystemDynamics(&system);


    // Create simulator
    teb::TebPlotter plotter;
    teb::Simulator<p,1> sim(&teb, system, &plotter);

    // Set simulation sample time
```

```
    sim.setSampleTime(0.1);

    // Simulate 12 seconds and save image to file (look in the folder of the binary)
    plotter.setOutputToFile("mpc_control_integrator",
      teb::TebPlotter::FileFormat::PNG);

    PRINT_INFO("Simulation running...");
    sim.simOpenAndClosedLoop(x0, xf, 12, false, false);
    PRINT_INFO("Simulation finished. Find the output file in your current working folder.");

    return 0;
}
```

18.4 integrator_system_mex.cpp

The following examples clarifies, how to prepare the TEB Controller (in particular the Controller of the Integrator-System example) for the usage in Matlab as a mex function.

The header file remains unchainged. File **integrator_system.h:**

```
#ifndef __teb_package__examples_integrator_system__
#define __teb_package__examples_integrator_system__


#include <teb_package.h>


template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
  using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
  using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector
      ;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      StateVector state_equations;

      const int no_int = p-1;
      if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

      // add last equation containing control u
      state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
      return state_equations;
  }


    void setTimeConstant(double time_constant) {_time_constant = time_constant;};

protected:

    double _time_constant = 1;
};


#endif /* defined(__teb_package__examples_integrator_system__) */
```

The following cpp-file shows how embed the TebController object into Matlab's function wrapper `mexFunction`. The example also wraps the mex function into a Matlab class. The controller class object is transformed to a Matlab handle using a small help (undocumented) helper class teb::matlab::matlabClassHandle(). See the Matlab class `control_interface.cpp` in *examples/matlab_interface* directory for more details on how to use it in Matlab.

An alternate approach without the helper class would be to store a persistent object (and its pointer) within the file (like for the Solver and System class). **Note**, the wrapper class can be easily adapted to other controllers by just changing the typedefs at the top of the file. Controller settings can be modified below the comment *// Setup Controller*.

File **integrator_system_mex.cpp**

```
#include "mex.h"
#include <iostream>

#include "integrator_system.h"
```

```
// support helper class to define a pointer suited for matlabs handle object
// @see teb::matlab::matlabClassHandle
using namespace teb::matlab;

// Create typedef for our controller, solver and system type
typedef teb::TebController<2, 1> ControllerType;
typedef teb::SolverLevenbergMarquardtEigenSparse SolverType;
typedef IntegratorSystem<2> SystemType;

// Create global variables
std::unique_ptr<SolverType> solver; // Solver
std::unique_ptr<SystemType> dynamics; // System dynamics

void mexFunction(int nlhs, mxArray *plhs[],
    int nrhs, const mxArray *prhs[])
{
    // Get command string
    char cmd[64];
    if (nrhs < 1 || mxGetString(prhs[0], cmd, sizeof(cmd)))
        mexErrMsgTxt("First input should be a command string with less than 64 characters.");


    // Create new obejct if command equals "new"
    if (!strcmp("new", cmd))
    {
        // Check parameters
        if (nlhs != 1)
            mexErrMsgTxt("New: One output expected.");


        // Instantiate new objects
        solver = std::unique_ptr<SolverType>(new SolverType);
        dynamics = std::unique_ptr<SystemType>(new SystemType);
        ControllerType* ctrl = new ControllerType(solver.get());

        // Setup controller
        ctrl->setSystemDynamics(dynamics.get()); // Satisfy system dynamics

        ctrl->activateObjectiveTimeOptimal(); // Minimum time control
        ctrl->activateControlBounds(0, -1.0, 1.0); // Control input bounds

        // Return handle to the new controller instance
        plhs[0] = convertPtr2Mat< ControllerType >(ctrl);

        return;
    }


    // For further work, we need a second input (class instance handle)
    if (nrhs < 2)
        mexErrMsgTxt("Second input should be a class instance handle.");

    // Delete object
    if (!strcmp("delete", cmd))
    {
        destroyObject<ControllerType>(prhs[1]);
        if (nlhs != 0 || nrhs != 2)
            mexWarnMsgTxt("Delete: Unexpected arguments ignored.");
        return;
    }

    // For further work, retrieve the class instance pointer from the second input
    ControllerType* ctrl = convertMat2Ptr<ControllerType>(prhs[1]);

    // Now start calling desired class method

    // Call teb::BaseController::step()
    if (!strcmp("step", cmd))
    {
        // Check inputs and outputs
        if (nrhs != 4)  mexErrMsgIdAndTxt("teb_package:step:nrhs", "Four inputs required: cmd, obj, x0 and
       xf [2x1].");
        if (nlhs != 1)  mexErrMsgIdAndTxt("teb_package:step:nlhs", "One output required: u [1x1]//"); //
       teb [3 x n_max], dt [1x1].");

        double* x0 = mxGetPr(prhs[2]); // start state
        double* xf = mxGetPr(prhs[3]); // goal state
        double u = 0; // store control

        // Perform open-loop planning
        ctrl->step(x0, xf, &u);

        // create output (control input u)
        plhs[0] = mxCreateDoubleScalar(u);
        return;
    }
```

```
    // Return TEB matrix teb::BaseController::getStateCtrlInfoMat()
    if (!strcmp("tebmatrix", cmd))
    {
        // Check inputs and outputs
        if (nrhs != 2) mexErrMsgTxt("No more input parameter allowed.");
        if (nlhs != 1) mexErrMsgTxt("One output for the teb matrix required.");

        // Size of the mat: [p+q+1 x n] // +1: add time vector
        int m = ctrl->NoStates + ctrl->NoControls + 1;
        int n = ctrl->getN();
        plhs[0] = mxCreateNumericMatrix(m, n, mxDOUBLE_CLASS, mxREAL);

        // Get values
        // Wrap mxArray into an Eigen::Matrix
        Eigen::Map<Eigen::Matrix<double, -1, -1, Eigen::ColMajor>> mat_map(mxGetPr(plhs[0]), m, n); //
       Matlab uses column major representation
        mat_map.topRows(1) = ctrl->getAbsoluteTimeVec().transpose();
        mat_map.bottomRows(m-1) = ctrl->getStateCtrlInfoMat();
        return;
    }


    // If this line is reached, the command is not recognized
    mexErrMsgTxt("Command noch recognized.");
}
```

18.5 integrator_system_sfun.cpp

The following examples clarifies, how to prepare the TEB Controller (in particular the Controller of the Integrator-System example) for the usage in Matlab Simulink as a S-Function block.

The header file remains unchainged. File **integrator_system.h:**

```
#ifndef __teb_package__examples_integrator_system__
#define __teb_package__examples_integrator_system__


#include <teb_package.h>


template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
  using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
  using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector
      ;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      StateVector state_equations;

      const int no_int = p-1;
      if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

      // add last equation containing control u
      state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
      return state_equations;
  }


    void setTimeConstant(double time_constant) {_time_constant = time_constant;};

protected:

    double _time_constant = 1;
};



#endif /* defined(__teb_package__examples_integrator_system__) */
```

The following cpp-file shows how embed the TebController object into Matlab's S-Function wrappers.

File **integrator_system_sfun.cpp**

```
#include "integrator_system.h"
```

```
#ifdef __cplusplus
extern "C" { // use the C fcn-call standard for all functions
#endif       // defined within this scope

#define S_FUNCTION_NAME integrator_system_sfun /* Defines and Includes */
#define S_FUNCTION_LEVEL 2

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"


#define MDL_INITIAL_SIZES
static void mdlInitializeSizes(SimStruct *S)
{

    // Reserve elements in the pointer work vector to store C++ objects
    // (see mdlStart function for object instantiation)
    ssSetNumPWork(S, 3); // Assume 3 objects (controller, solver, system)

    // Check inputs and outputs
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch reported by the Simulink engine*/
    }

    if (!ssSetNumInputPorts(S, 2)) return;
    // input 1: xf
    ssSetInputPortWidth(S, 0, 2);
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    // input 2: x0
    ssSetInputPortWidth(S, 1, 2);
    ssSetInputPortDirectFeedThrough(S, 1, 1);

    // output: u
    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, 1);

    ssSetNumSampleTimes(S, 1); // TODO

    /* Take care when specifying exception free code - see sfuntmpl.doc */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
    }

#define MDL_INITIALIZE_SAMPLE_TIMES
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME); // TODO
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_START
static void mdlStart(SimStruct *S)
{
    // Instantiate persistent objects (delete objects in mdlTerminate)
    teb::BaseSolver* solver = new
      teb::SolverLevenbergMarquardtEigenSparse;
    teb::TebController<2, 1>* ctrl = new
      teb::TebController<2,1>(solver);
    IntegratorSystem<2>* int_sys = new IntegratorSystem<2>;

    // Store void pointers in simulinks work vector for persistent variables.
    ssGetPWork(S)[0] = (void*) solver; // solver
    ssGetPWork(S)[1] = (void*) ctrl; // controller
    ssGetPWork(S)[2] = (void*) int_sys; // system

    // setup controller
    ctrl->setSystemDynamics(int_sys);
    ctrl->activateObjectiveTimeOptimal();
    ctrl->activateControlBounds(0, -1.0, 1.0);
}

#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    // reset trajectroy
    teb::TebController<2, 1>* ctrl = (
      teb::TebController<2, 1>*) ssGetPWork(S)[1];
    ctrl->resetController();
}

#define MDL_OUTPUTS
static void mdlOutputs(SimStruct *S, int_T tid)
```

```
{
    // Retrieve persistent controller object
    teb::TebController<2, 1>* ctrl = (
      teb::TebController<2, 1>*) ssGetPWork(S)[1];

    // Get input and output parameters
    InputRealPtrsType xfPtrs = ssGetInputPortRealSignalPtrs(S,0);
    InputRealPtrsType x0Ptrs = ssGetInputPortRealSignalPtrs(S,1);
    real_T *uPtr = ssGetOutputPortRealSignal(S,0);

    // Perform MPC step
    ctrl->step(*x0Ptrs, *xfPtrs, uPtr);

}

static void mdlTerminate(SimStruct *S)
{
    // Destroy persistent objects
    teb::BaseSolver* solver = (teb::BaseSolver*) ssGetPWork(S)[0];
    teb::TebController<2, 1>* ctrl = (
      teb::TebController<2, 1>*) ssGetPWork(S)[1];
    IntegratorSystem<2>* int_sys = (IntegratorSystem<2>*) ssGetPWork(S)[2];
    delete solver;
    delete ctrl;
    delete int_sys;
}

#ifdef MATLAB_MEX_FILE /* Is this file being compiled as a MEX-file? */
#include "simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfun.h" /* Code generation registration function */
#endif

#ifdef __cplusplus
} // end of extern "C" scope
#endif
```

18.6 linear_system_ode.cpp

The following example shows the TEB controller applied to a common linear system described by an ordinary differential equation of the p -th order:

$$a_p x^{(p)}(t) + a_{p-1} x^{(p-1)}(t) + \ldots + a_1 \dot{x}(t) + a_0 x(t) = bu(t)$$

The control input u should be bounded to the interval [-1, 1].

Let $p = 2$, $b = 1$ and $\mathbf{a} = [a_2, a_1, a_0] = [0.8, 0.9, 1.0]$ that leads to the following ode:

$$0.8\ddot{x}(t) + 0.9\dot{x}(t) + 1.0x(t) = u(t)$$

The state vector is defined by $\mathbf{x} = [x, \dot{x}]^T$.

The objective of the control task is to transite the system from the start state $\mathbf{x}_s = [0, 0]^T$ to the final state $\mathbf{x}_f = [1, 0]^T$.

The system dynamics are specified using the LinearSystemODE class. See the following header file how to add the state space model mentioned above.

File **linear_system_ode.h:**

```
#ifndef __teb_package__examples_linear_system_ode__
#define __teb_package__examples_linear_system_ode__


#include <teb_package.h>


template <int p>
class LinearSystemODE : public teb::SystemDynamics<p,1>
{
public:
  using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
  using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector
      ;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
```

```
        StateVector state_equations;

        const int no_int = p-1;
        // integrator states -> control normal form
        if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

        // add last equation containing ODE coefficients
        state_equations[no_int] = _coeff_b*u[0]; // u
        for (int i = 0; i < p; ++i) // add all other ode coefficients
        {
            state_equations[no_int] -= _coeff_a[p-i] * x[i];
        }
        state_equations[no_int] /= _coeff_a[0];
        return state_equations;
    }

void setODECoefficients(const double* coeff_a, double coeff_b)
{
    _coeff_a = coeff_a;
    _coeff_b = coeff_b;
}

protected:
    const double* _coeff_a = nullptr;
    double _coeff_b = 1;
};


#endif /* defined(__teb_package__examples_linear_system_ode__) */
```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the LinearSystemODE object specified in linear_system_ode.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows:

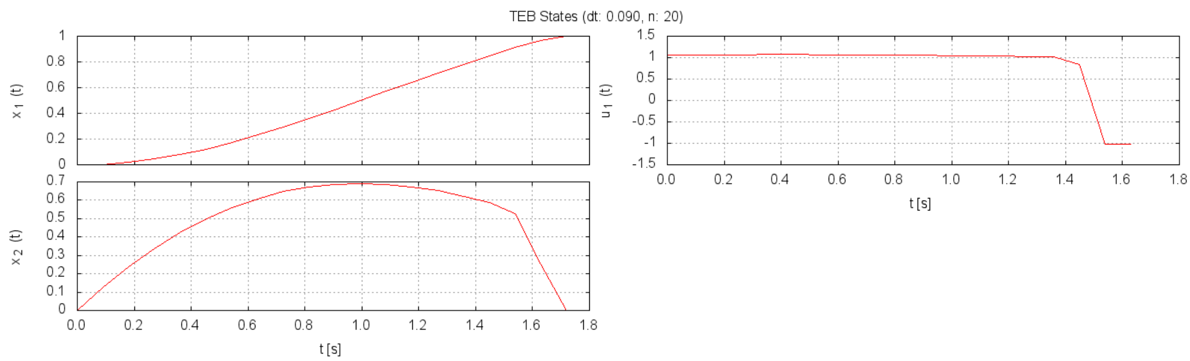


Figure 18.4: Result of linear_system_ode.cpp

File **linear_system_ode.cpp**

```
#include "linear_system_ode.h"


int main()
{

    const int p = 2; // number of states
    // ode: a_2 * xddot + a_1 * xdot + a_0 * x = b * u;
    const double coeff_a[] {0.8,0.9,1}; // a_2, a_1, a_0
    const double coeff_b  = 1; // no derivatives of the input supported atm.

    double x0[] = {0,0}; // start state
    double xf[] = {1,0}; // goal state
    double u; // store control

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&solver);
```

```
    // System specific settings
    LinearSystemODE<p> system;
    system.setODECoefficients(coeff_a, coeff_b);

    teb.setSystemDynamics(&system);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);

    // Perform open-loop planning
    START_TIMER;
    teb.step(x0, xf, &u);
    STOP_TIMER("TEB optimization loop");

    // Print determined control for the current sampling interval
    PRINT_INFO("Control u: " << u);

    // Plot states and control input
    teb::TebPlotter plotter;
    plotter.plotTEB(teb);

    return 0;
}
```

18.7 linear_system_ode_ctrl_comparison.cpp

The following example compares the time-optimal TEB controller with the common quadratic form MPC.

The controlled system is specified using an (unstable) ordinary differential equation (refer to other examples for more details).

The system dynamics are specified using the LinearSystemODE class. See the following header file how to add the state space model mentioned above.

File **linear_system_ode.h:**

```
#ifndef __teb_package__examples_linear_system_ode__
#define __teb_package__examples_linear_system_ode__


#include <teb_package.h>


template <int p>
class LinearSystemODE : public teb::SystemDynamics<p,1>
{
public:
  using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
  using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector
      ;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      StateVector state_equations;

      const int no_int = p-1;
      // integrator states -> control normal form
      if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

      // add last equation containing ODE coefficients
      state_equations[no_int] = _coeff_b*u[0]; // u
      for (int i = 0; i < p; ++i) // add all other ode coefficients
      {
          state_equations[no_int] -= _coeff_a[p-i] * x[i];
      }
      state_equations[no_int] /= _coeff_a[0];
      return state_equations;
  }

void setODECoefficients(const double* coeff_a, double coeff_b)
{
    _coeff_a = coeff_a;
    _coeff_b = coeff_b;
}

protected:
    const double* _coeff_a = nullptr;
    double _coeff_b = 1;
```

```
};


#endif /* defined(__teb_package__examples_linear_system_ode__) */
```

The following cpp-file shows how to create both controllers, how to add control bounds and how to add the Linear-SystemODE object specified in linear_system_ode.h.

The output of the program is as follows (closed-loop results only):

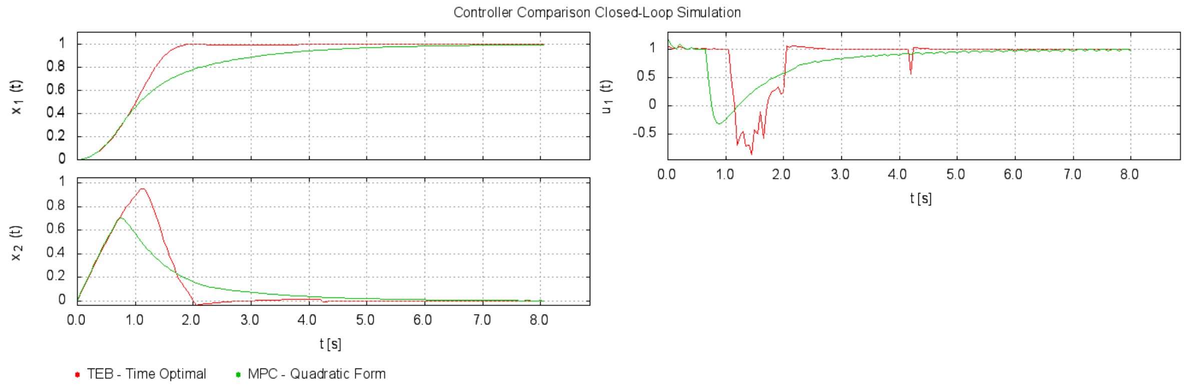


Figure 18.5: Result of linear_system_ode_ctrl_comparison.png

File **linear_system_ode_ctrl_comparison.cpp**

```
#include "linear_system_ode.h"


int main()
{

    const int p = 2; // number of states
    // ode: a_2 * xddot + a_1 * xdot + a_0 * x = b * u;
    const double coeff_a[] {1, -0.1, 1}; // a_2, a_1, a_0
    const double coeff_b  = 1; // no derivatives of the input supported atm.

    double x0[] = {0, 0}; // start state
    double xf[] = {1, 0}; // goal state

    // System specific settings
    LinearSystemODE<p> system;
    system.setODECoefficients(coeff_a, coeff_b);


    // Setup controller 1
    teb::SolverLevenbergMarquardtEigenSparse solver1;
    teb::TebController<p,1> teb(&solver1);

    teb.setSystemDynamics(&system);
    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);



    // Setup controller 2
    teb::Config mpc_cfg;
    mpc_cfg.optim.solver.lsq.weight_equalities = 4;
    teb::SolverLevenbergMarquardtEigenSparse solver2;
    teb::TebController<p,1> mpc(&mpc_cfg, &solver2);

    mpc.setSystemDynamics(&system);

    // Setup receding horizon T = n * dt  = 10 * 0.1
    // Common setting: unfixed goal and fixed resolution
    mpc.setupHorizon(false, true, 10, 0.1);

    mpc.activateObjectiveQuadraticForm(1, 0.1, 100); // Parameters: Q, R, Qf
    mpc.activateControlBounds(0, -1.0, 1.0);
```

```
    // Create simulator
    teb::TebPlotter plotter;
    teb::Simulator<p,1> sim(system,&plotter);
    // Set simulation sample time
    sim.setSampleTime(0.05);

    //sim.setIntegrator(teb::NumericalIntegrators::EXPLICIT_EULER);

    // Create container storing the controllers that should be compared
    std::vector<std::pair<teb::BaseController*, std::string>> controllers;
    controllers.emplace_back(&teb, "TEB - Time Optimal");
    controllers.emplace_back(&mpc, "MPC - Quadratic Form");

    // Closed-loop sim
    sim.simClosedLoop(controllers, x0, xf, 8);

    // Open-loop sim
    // create new figure with id 1
    plotter.switchWindow(1,true);
    sim.simOpenLoop(controllers, x0, xf);

    return 0;
}
```

18.8 linear_system_state_space.cpp

The following example shows the TEB controller applied to a common linear system described by a continuous-time state space model of the p -th order:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \qquad \mathbf{x} \in \mathbb{R}^p,\ \mathbf{u} \in \mathbb{R}^q$$

For this example, let $p = 2$ and $q = 1$. We define $\mathbf{A} = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 1 \end{smallmatrix}\right]$ and $\mathbf{b} = [0,1]^T$.

The control input u should be bounded to the interval [-1, 1].

The objective of the control task is to transite the system from the start state $\mathbf{x}_s = [0,0]^T$ to the final state $\mathbf{x}_f = [1,0]^T$.

The system dynamics are specified using the LinearSystemStateSpace class. See the following header file how to add the state space model mentioned above.

File **linear_system_state_space.h:**

```
#ifndef __teb_package__examples_linear_system_state_space__
#define __teb_package__examples_linear_system_state_space__


#include <teb_package.h>


template <int p, int q>
class LinearSystemStateSpace : public teb::SystemDynamics<p,q>
{
public:
  using StateVector = typename teb::SystemDynamics<p,q>::StateVector;
  using ControlVector = typename teb::SystemDynamics<p,q>::ControlVector
      ;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      return _A * x + _B * u;
  }

  void setStateSpaceModel(const Eigen::Ref<const Eigen::Matrix<double,p,p>>& A, const Eigen::Ref<const
      Eigen::Matrix<double,p,q>>& B)
  {
    _A = A;
    _B = B;
  }

protected:
    Eigen::Matrix<double,p,p> _A;
    Eigen::Matrix<double,p,q> _B;
};


#endif /* defined(__teb_package__examples_linear_system_state_space__) */
```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the LinearSystemStateSpace object specified in linear_system_state_space.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows:

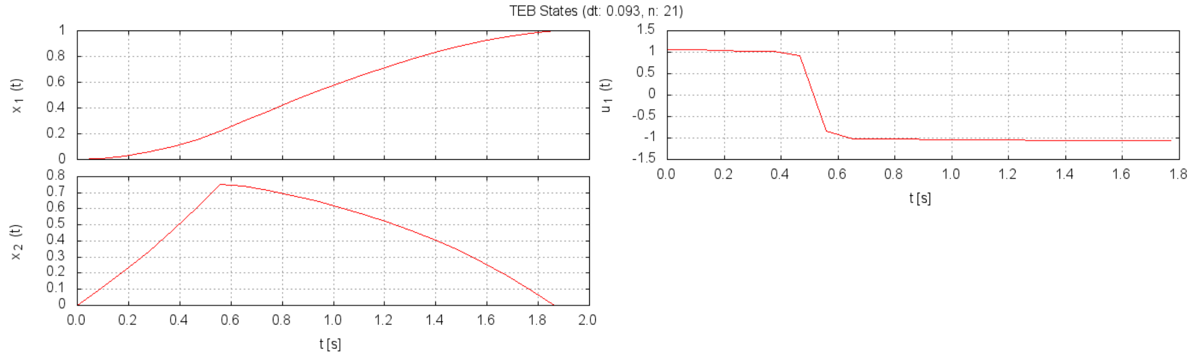


Figure 18.6: Result of linear_system_state_space.cpp

File **linear_system_state_space.cpp**

```
#include "linear_system_state_space.h"


int main()
{

    const int p = 2;
    const int q = 1;

    Eigen::Matrix2d A;
    A(0,0) = 0;
    A(0,1) = 1;
    A(1,0) = 0;
    A(1,1) = 1;

    Eigen::Vector2d b;
    b(0) = 0;
    b(1) = 1;

    double x0[] = {0,0};
    double xf[] = {1,0};
    double u;

    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&solver);

    LinearSystemStateSpace<p,q> system;
    system.setStateSpaceModel(A,b);

    teb.setSystemDynamics(&system);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);

    START_TIMER;
    teb.step(x0, xf, &u);
    STOP_TIMER("TEB optimization loop");

    PRINT_INFO("Control u: " << u);

    teb::TebPlotter plotter;
    plotter.plotTEB(teb);

    return 0;
}
```

18.9 mobile_robot_teb.cpp

This example shows how to build a customized controller that minimizes a used-defined optimization problem rather than using default optimization functions.

Part of the example is a mobile robot trajectory planning and control application. The optimization problem is given by the original timed-elastic-band presented in [3].

For the sake of simplicity, we ignore some optional cost-functions like constraints on accelerations and the forward drive direction. In addition, we apply only a uniform ΔT.

We first start by introducing the optimization problem. The robot pose is defined by $\mathbf{x} = [x, y, \beta]^T$. The robot trajectory is than given by the sequence of $n \in \mathbb{N}$ poses $\mathbf{x}_i, i = 1, \ldots, n$. Between two consecutive poses $\{\mathbf{x}_i, \mathbf{x}_{i+1}\}$ we define the time difference ΔT, that the robot requires to transit from the a configuration to the subsequent one in the sequence.

The robot hardware interface accepts translational and rotational velocities $\mathbf{u} = [v, \omega]^T$ as control inputs.

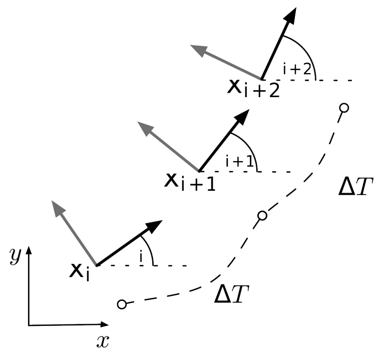


Figure 18.7: Mobile robot trajectory representation

In conrast to common MPC approaches, the sequence of poses and the time difference are part of the optimzation rather than the control inputs $\mathbf{u}$, because they are implicitly part of the pose sequence and they can be derived via difference quotients.

The joint trajectory respentation is than defined as follows:

$$\mathscr{B} := \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1}, \mathbf{x}_n, \Delta T\}$$

In the following we construct the optimization problem.

A general cost function can often be decomposed into the sum of z "local" cost terms:

$$V(\mathscr{B}) = f(\mathscr{B}) = \sum_i^z f_i(\mathscr{B}_i)$$

f_i denotes a local cost term and $\mathscr{B}_i \subseteq \mathscr{B}, i = 1, \ldots, z$ denotes a (small) subset of $\mathscr{B}$.

Obviously, if the subset of depending optimization variables is low for each local cost term, the resuling optimization problem has a sparse structure. To exploit this structure later, a hyper-graph (like mentioned) is ideally suited to capture this property already during the problem formulation phase.

Optimization problems in this package are formulated as hyper-graphs (according to [2]) with additional distinction between constraints and objective functions. A hyper-graph is a graph, in which edges connect an arbitary number of vertices (not just two). In sense of optimization, edges represent local cost function terms $f_k(\mathscr{B}_i)$ and vertices represent the optimization variables (in this package commonly $\mathbf{x}_k, \mathbf{u}_k, \Delta T$). Consequently for the mobile robot problem, vertices are: $\mathbf{x}_k$ and ΔT (see the image a few lines below for an example hyper-graph).

The considered mobile robot has two wheels connected with a differential drive. The formulation of the optimization problem in [3] does not take a common robot motion model into account. The kinodynamic motion behavior (constraints on robot / wheel velocities / accelerations and the non-holonomic kinematic) is captured using the superposition of dedicated cost functions for each objective or constraint. For the sake of simplicty we reduce the optimization task to just limit velocities of the center of the robot, to satisfy non-holonomic kinematics and to avoid obstacles:

$$V(\mathscr{B}) = f(\mathscr{B}) = f_{mintime}(\Delta T) + \sum_{i=1}^{n-1} [f_{i,vel}(\mathscr{B}) + f_{i,kin}(\mathscr{B}) + f_{i,obst}(\mathscr{B})]$$

Each cost term above is a squared function and hence the optimization problem can be solved using the Levenberg-Marquardt algorithm. Bounding the velocity, satisfying a minimum distance to obstacles and satisfying the kinematic constraints are implemented using soft-constraints in [**?**].

Within this MPC package, we have the oppertunity to specify constraints in a more genalized way. And if the selected solver is a least-squares solver (like teb::SolverLevenbergMarquardtEigenSparse), than the soft-constraint approach is applied in similar way like in [3]. As a result, we can generalize the above optimization problem for the mobile robot to:

$$V^*(\mathscr{B}) = \min f_{mintime}(\Delta T) \tag{18.1}$$

$$\text{s.t.:}$$

$$\mathbf{x}_1 = \mathbf{x}_s \tag{18.2}$$

$$\mathbf{x}_n = \mathbf{x}_f \tag{18.3}$$

$$\mathbf{h}_{i,kin}(\mathscr{B}) = \mathbf{0} \quad i \in 1,\dots,n-1 \tag{18.4}$$

$$\mathbf{g}_{i,vel}(\mathscr{B}) \leq \mathbf{0} \quad i \in 1,\dots,n-1 \tag{18.5}$$

$$\mathbf{g}_{i,obst}(\mathscr{B}) \leq \mathbf{0} \quad i \in 1,\dots,n-1 \tag{18.6}$$

$\mathbf{h}$ and $\mathbf{g}$ denote the equality and inequality constraints without squared soft-constraint approximations.

According to [3] the cost terms are as follows (**without soft-constraint approx** and **without taking squared**):

- Minimum transition time: $f_{mintime}(\Delta T) = (n-1)\Delta T$ (Note, least-square solvers square all objectives later!)
- Limit velocity:

$$\begin{aligned}
\Delta\mathbf{s} &= \begin{bmatrix} \left\| \begin{bmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \end{bmatrix} \right\| \\ \text{normAngle}(\beta_{i+1} - \beta_i) \end{bmatrix} \\
\mathbf{g}_{i,vel}(\mathscr{B}) &= \frac{1}{\Delta T} \begin{bmatrix} \Delta\mathbf{s} \\ -\Delta\mathbf{s} \end{bmatrix} + \begin{bmatrix} -v_{max} \\ -\omega_{max} \\ v_{min} \\ \omega_{min} \end{bmatrix}
\end{aligned}$$

- Nonholonomic Kinematics:

$$\begin{aligned}
\mathbf{d}_i &= \begin{bmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \\ 0 \end{bmatrix} \\
\mathbf{h}_{i,kin} &= \left\| \left[\begin{bmatrix} \cos\beta_i \\ \sin\beta_i \\ 0 \end{bmatrix} + \begin{bmatrix} \cos\beta_{i+1} \\ \sin\beta_{i+1} \\ 0 \end{bmatrix} \right] \times \mathbf{d}_i \right\|
\end{aligned}$$

- Obstacle avoidance (State constraint):

$$\mathbf{g}_{i,obst} = -\left\| \begin{bmatrix} x_i - x_{obst} \\ y_i - y_{obst} \end{bmatrix} \right\| + d_{min}$$

Please note, that using a least-squares soft-constrained solver (as mentioned above), all constraints and objectives are incorporated in a weighted objective/cost function and are becoming squared. The above optimization problem can easily be transfered to a hyper-graph. See the following figure for the resulting example graph. All indices are increased by one subsequently per constraint/objective type.

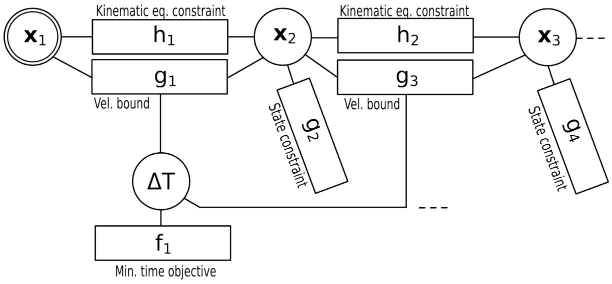


Figure 18.8: Example hyper-graph of the mobile robot application

The first state $\mathbf{x}_1$ is marked with a double circle. That means the state is fixed during optimization (to define the current/initial state). The goal state is fixed as well (according to a fixed horizon). Fixing these values implicitly satisfies $\mathbf{x}_1 = \mathbf{x}_s$ and $\mathbf{x}_n = \mathbf{x}_f$ of the problem definition above.

In contrast to [3] we are using a single ΔT that is part of all velocity edges (note the connection of $\mathbf{g}_1, \mathbf{g}_3$ and ΔT in the example graph. Formulating the optimization problem as a hyper-graph immediately indicates the structure of the underlying hessian/jabian matrix. For example, considering the ordering of the optimization variables of $\mathscr{B}$, than the hessian of the Lagrangian or the hessian of a least-square cost-function with soft-constraints would contain a band diagonal (since the maximum number of connected states $\mathbf{x}_i$ is at least two. Only the velocity bound edges that include always the same ΔT would lead to a dense last row and column in the matrix.

The following header-file demonstrates how to implement the "local" cost functions mentioned above (please refer to teb::BaseEdge for an overview of possible edges to derive from):

File **rob_cost_edges.h:**

```
#ifndef __teb_package__examples_rob_cost_edges__
#define __teb_package__examples_rob_cost_edges__


#include <teb_package.h>


namespace teb
{

    // Declarations for cost functions / hyper edges

    // Non-holonomic-Kinematics
    class EdgeKinematics : public BaseEdge<1, FUNCT_TYPE::NONLINEAR, StateVertex<3>, StateVertex<3>> //
       Cost vector dimension: 1, Vertices: StateVertex
    {
    public:

        EdgeKinematics(StateVertex<3>& state1, StateVertex<3>& state2) : BaseEdge<1, FUNCT_TYPE::NONLINEAR,
       StateVertex<3>, StateVertex<3>>(state1, state2) {}

        virtual void computeValues() // See EdgeType::computeValues()
        {
            StateVertex<3>* sample_i = static_cast<StateVertex<3>*>(_vertices[0]);
            StateVertex<3>* sample_ip1 = static_cast<StateVertex<3>*>(_vertices[1]);

            // Direction vector between two samples:
            Eigen::Vector2d deltaS = sample_ip1->states().head(2) - sample_i->states().head(2);
            double angle_i = sample_i->states()[2];
            double angle_ip1 = sample_ip1->states()[2];

            // Nonholonomic Kinematics
            _values[0] = fabs( ( cos(angle_i)+cos(angle_ip1) ) * deltaS[1] - ( sin(angle_i)+sin(angle_ip1)
      ) * deltaS[0] );
        }

        // Hessian and Jacobians are calculated numerically, otherwise add implementation here.


        EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    };

    // Limit Velocity
    class EdgeLimitVelocity : public BaseEdge<4, FUNCT_TYPE::NONLINEAR, StateVertex<3>, StateVertex<3>, Tim
      eDiff> // Cost vector dimension: 4, Vertices: StateVertex, StateVertex, TimeDiff
    {
```

```
    public:

        EdgeLimitVelocity(StateVertex<3>& state1, StateVertex<3>& state2, TimeDiff& dt) : BaseEdge<4,
       FUNCT_TYPE::NONLINEAR, StateVertex<3>, StateVertex<3>, TimeDiff>(state1, state2, dt)
        {
        }

        virtual void computeValues() // See EdgeType::computeValues()
        {
            StateVertex<3>* sample_i = static_cast<StateVertex<3>*>(_vertices[0]);
            StateVertex<3>* sample_ip1 = static_cast<StateVertex<3>*>(_vertices[1]);
            TimeDiff* dt = static_cast<TimeDiff*>(_vertices[2]);

            // Direction vector between two samples:
            Eigen::Vector2d deltaS = sample_ip1->states().head(2) - sample_i->states().head(2);
            double vel = deltaS.norm() / dt->dt();
            double omega = norm_angle( sample_ip1->states()[2] - sample_i->states()[2] ) / dt->dt
      ();

            // Velocity limits: c(x) < 0
            _values[0] = vel - v_max;
            _values[1] = -vel + v_min;
            _values[2] = omega - omega_max;
            _values[3] = -omega + omega_min;
        }

        // Hessian and Jacobians are calculated numerically, otherwise add implementation here.

        double v_max = 1;
        double v_min = -1;
        double omega_max = 0.5;
        double omega_min = -0.5;

        EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    };


    // Obstacle-Avoidance (point-representation)
    class EdgeObstacle : public BaseEdge<1, FUNCT_TYPE::NONLINEAR, StateVertex<3>> // Cost vector
       dimension: 1, Vertices: StateVertex
    {
    public:

        EdgeObstacle(StateVertex<3>& state) : BaseEdge<1, FUNCT_TYPE::NONLINEAR, StateVertex<3>>(state) {}

        virtual void computeValues() // See EdgeType::computeValues()
        {
            StateVertex<3>* sample_i = static_cast<StateVertex<3>*>(_vertices[0]);

            // Direction vector between sample and obstacle:
            Eigen::Vector2d deltaS = sample_i->states().head(2) - obstacle_position;
            //double angle_i = sample_i->states()[2];

            // Projection of the distance deltaS orthogonal to the TEB
        //double angdiff = atan2(deltaS[1],deltaS[0])-angle_i;
        double proj_dist = deltaS.norm();//*fabs(sin(angdiff));

        // Constraint proj_dist > min_distance -> -proj_dist + min_distance < 0
            _values[0] = -proj_dist + min_distance;
        }

        // Hessian and Jacobians are calculated numerically, otherwise add implementation here.


        Eigen::Vector2d obstacle_position; // DO NOT FORGET TO SET VALUE
        double min_distance = 0.5;

        EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    };


} // end namespace teb


#endif /* defined(__teb_package__examples_rob_cost_edges__) */
```

In order to add the hyper-graph to the TebController, you can derive the class teb::TebController an override the method teb::TebController::customOptimizationGraph(). This ensures that the default edges like teb::EdgeMinimizeTime or teb::EdgeQuadraticForm can be added as well using the controller API. You can change states to fixed or unfixed as well. (Please note, that the default fixing and unfixing of the start and goal state as well as the default edges for the hyper-graph are applied in teb::TebController::buildOptimizationGraph().

The derived class for the mobile-robot example is defined as follows. (**Note**, since the problem is defined with only implicit control inputs, by means of q=0, the teb::TebController::firstControl() and teb::TebController::returnControlInputSequence() are overridden as well. Additionally, we extend the new controller API with more debug and visualization functions.

File **rob_controller.h:**

```
#ifndef __teb_package__examples_rob_controller__
#define __teb_package__examples_rob_controller__


#include <teb_package.h>
#include "rob_cost_edges.h"


namespace teb
{

    // Controller implementation
    class RobController : public TebController<3,0>
    {

    public:

#ifdef WIN32 // VS2013 does not support constructor inheritance (only starting from Nov 2013 CTP)
        RobController(BaseSolver* solver = nullptr) : TebController(solver) {};
        RobController(const Config* config, BaseSolver* solver = nullptr) : TebController(config, solver) {
      };
#else
        using TebController<3,0>::TebController; // inherite base constructors
#endif

        // Overload customOptimizationGraph() to define a custom hyper-graph
        // and thus preserve the option to add predifined edges like time-optimality
        virtual void customOptimizationGraph()
        {

            for (unsigned int i=0; i<getN()-1; ++i)
            {
                // Add non-holonomic kinematics
                EdgeKinematics* kinematics_edge = new EdgeKinematics(_state_seq.at(i), _state_seq.at(i + 1)
      );
                _graph.addEdgeEquality(kinematics_edge);

                // Add velocity limits (translational and rotational)
                EdgeLimitVelocity* vel_edge = new EdgeLimitVelocity(_state_seq.at(i), _state_seq.at(i + 1),
       _dt);
                _graph.addEdgeInequality(vel_edge);

                // Add single obstacle edge
                EdgeObstacle* obst_edge = new EdgeObstacle(_state_seq.at(i));
                obst_edge->min_distance = 0.5;
                obst_edge->obstacle_position = _obstacle_pos;
                _graph.addEdgeInequality(obst_edge);
            }


        };

     void setObstaclePosition(const Eigen::Ref<const Eigen::Vector2d>& obstacle_pos) {_obstacle_pos =
      obstacle_pos;};
     void setObstaclePosition(const double* obstacle_pos) {_obstacle_pos = Eigen::Vector2d(obstacle_pos);};

     void setCurrentVelocity(const Eigen::Ref<const Eigen::Vector2d>& curr_velocity) {_curr_velocity =
      curr_velocity;};
     void setCurrentVelocity(double v, double omega) {_curr_velocity[0]=v; _curr_velocity[1]=omega;};

     // Get all velocities (implicit from states)
     // This method is not efficient, but it is only used for visualization here.
     // Returns a [2 x n-1] matrix in which each column denotes: [sqrt(vx^2+vy^2), omega]^T
     Eigen::MatrixXd getVelocities()
     {
          unsigned int n = getN(); // Number of states
          Eigen::MatrixXd teb_mat = getStateCtrlInfoMat(); // inefficient here, but comfortable
          Eigen::Matrix<double,3,-1> vel3 = (teb_mat.topRightCorner(3,n-1) - teb_mat.topLeftCorner(3,n-1));
          // Normalize angle differnece
          norm_angle_vec(vel3.bottomRows(0));
          vel3 /= dt().dt(); // scale to actual velocities

          Eigen::MatrixXd vel2(2,vel3.cols());
          vel2.row(0) = vel3.topRows(2).colwise().norm();
          vel2.row(1) = vel3.row(2);
          return vel2;
     }
```

```
// Override original function to return the current control input, since
// we use an implicit control input model (as a function of the states)
Eigen::VectorXd firstControl() const
{
    Eigen::VectorXd u(2,1);
    u[0] = (_state_seq.at(1).states().head(2) - _state_seq.at(0).states().head(2)).norm() / dt().dt();
  // transl. velocity
    u[1] = norm_angle((_state_seq.at(1).states()[2] - _state_seq.at(0).states()[2])) / dt().
 dt(); // rot. velocity

// Check if Goal reached:
if ( (firstStateRef() - lastStateRef() ).norm() < 0.1)
{
  u.setZero();
  PRINT_INFO_ONCE("RobController: Goal Reached.");
}

    return u;
}

// Override this function as well to get all implicit controls at once
virtual Eigen::MatrixXd returnControlInputSequence() const
{
    Eigen::MatrixXd states = getStateCtrlInfoMat();

    unsigned int m = getN()-1;
    Eigen::MatrixXd u_seq(2,m);

    u_seq.row(0) = ( states.topRightCorner(2,m) - states.topLeftCorner(2,m) ).colwise().norm() / getDt
 ();
    u_seq.row(1) = states.bottomRightCorner(1, m) - states.bottomLeftCorner(1, m);
    norm_angle_vec(u_seq.bottomRows(0));
    u_seq.row(1) /= getDt();

    return u_seq;
}

 // Update current velocity after each step to initialize the subsequent step
 void robSaveVelocityAfterStep(BaseController* ctrl, SystemDynamics<3,2>* system, Simulator<3,2>* sim)
 {
setCurrentVelocity(firstControl());
 }



 // We need some additional visualization
 void plotRobotStuff()
 {
// Plot states and control input
TebPlotter plotter;
plotter.plotTEB(*this);

// Create x-y plot
plotter.switchWindow(1,true); // Create new window
Eigen::MatrixXd teb_mat = getStateCtrlInfoMat(); // Get all TEB states in a single matrix
plotter.plot(teb_mat.row(0).transpose(), teb_mat.row(1).transpose(),"X-Y Trajectory","x [m]", "y [m]");

// Get velocity from the states
Eigen::MatrixXd vel = getVelocities();

// plot resulting translational velocity
plotter.switchWindow(2,true);

PlotOptions options;
options.title = "Robot velocities";
options.ylabels.emplace_back("trans. vel [m/s]");
options.ylabels.emplace_back("rot. vel [rad/s]");

plotter.plotMulti(getAbsoluteTimeVec().head(getN()-1), vel, &options);
 }

protected:
 Eigen::Vector2d _curr_velocity = Eigen::Vector2d::Zero(); // [v, omega]^T
 Eigen::Vector2d _obstacle_pos = Eigen::Vector2d::Zero();

};



// Velocity bounds
```

```
} // end namespace teb


#endif /* defined(__teb_package__examples_rob_controller__) */
```

Furthermore, for simulation purposes we define a teb::SystemDynamics model with simple integrator dynamics and the non-holonomic kinematics:

File **mobile_robot_teb.h:**

```
#ifndef __teb_package__examples_mobile_robot_teb__
#define __teb_package__examples_mobile_robot_teb__


#include <teb_package.h>

#include "rob_controller.h" // include customized controller


// Specify system model (here for simulation only, but could be used for the controller as well (as an
       alternative))
// We use a simple kinematic model here with integrator dynamics
class MobileRobot : public teb::SystemDynamics<3,2>
{
public:
  using StateVector = teb::SystemDynamics<3,2>::StateVector;
  using ControlVector = teb::SystemDynamics<3,2>::ControlVector;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      StateVector state_equations;

      // kinematic motion constraints
      // states = [x y theta]
      // inputs = [v, omega]

      state_equations[0] = u[0] * std::cos(x[2]); // xdot = v*cos(theta)
      state_equations[1] = u[0] * std::sin(x[2]); // ydot = v*sin(theta)
      state_equations[2] = norm_angle(u[1]); // thetadot = omega
      return state_equations;
  }

};


#endif /* defined(__teb_package__examples_mobile_robot_teb__) */
```

Finally, the main function that creates all objects, defines the optimization weights an initializes the simulation can be found in **mobile_robot_teb.cpp** (see below at the end).

The output of the program is as follows (the **obstacle** is placed at $x_{obst} = 2.5\,\mathrm{m}$ and $y_{obst} = 0.1\,\mathrm{m}$):

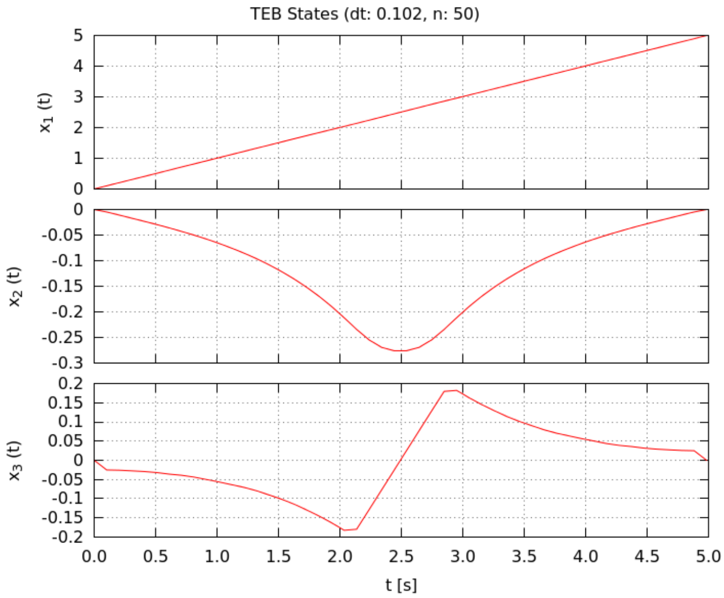


Figure 18.9: TEB States after the first open-loop optimization

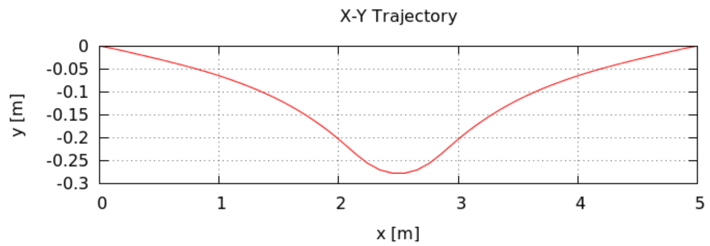


Figure 18.10: X-Y Trajectory after the first open-loop optimization

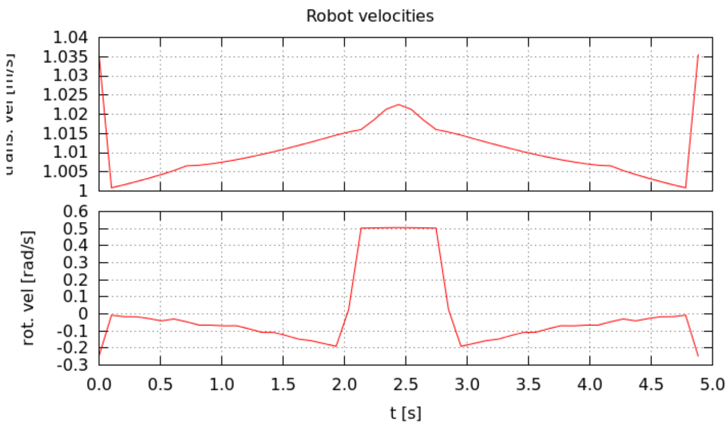


Figure 18.11: Velocity profile after the first open-loop optimization

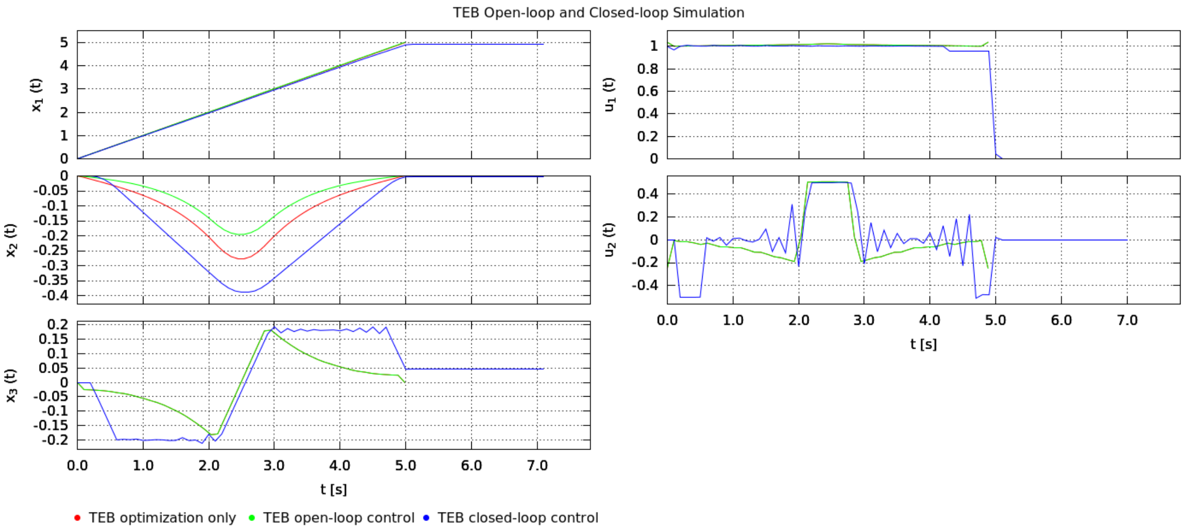


Figure 18.12: Closed-loop simulation of the mobile robot

File **mobile_robot_teb.cpp:**

```
#include "mobile_robot_teb.h"


int main()
{

    teb::Config cfg;
    cfg.optim.solver.lsq.weight_equalities = 500;
    cfg.optim.solver.lsq.weight_inequalities = 50;
    cfg.optim.solver.lsq.weight_adaptation_factor = 2;
    cfg.teb.dt_min = 0.05;
    cfg.teb.n_pre = 50;
    cfg.teb.n_max = 150;
    cfg.teb.n_min = 2;

    double x0[] = {0,0, 0};
    double xf[] = {5,0, 0};
    double obst_pos[] = {2.5, 0.1};

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::RobController teb(&cfg, &solver);

    teb.activateObjectiveTimeOptimal();
    teb.setObstaclePosition(obst_pos);

    // Perform open-loop planning
    START_TIMER;
    teb.step(x0, xf);
    STOP_TIMER("TEB optimization loop");

    // Plot open-loop results
    teb.plotRobotStuff();

    // Perform closed-loop sim
    teb::TebPlotter plotter;
    MobileRobot robot; // Motion model for simulation
    teb::Simulator<3,2> sim(&teb,robot,&plotter);

    // Set simulation sample time
    sim.setSampleTime(0.1);

    using namespace std::placeholders;
    sim.setPostStepCallback(std::bind(&teb::RobController::robSaveVelocityAfterStep, &
      teb, _1, _2, _3));

    // Start simulation
    sim.simOpenAndClosedLoop(x0,xf,7);

    return 0;
}
```

18.10 rocket_system.cpp

The following example shows the TEB controller applied to a simple free space rocket model with three differential states *s*, *v* and *m*.(http://acado.sourceforge.net/doc/html/d9/d65/example_001.html)

$$\dot{s}(t) = v(t) \quad (18.7)$$

$$\dot{v}(t) = \frac{u(t) - 0.02v^2(t)}{m(t)} \quad (18.8)$$

$$\dot{m}(t) = -0.01u^2(t) \quad (18.9)$$

The control input u should be bounded to the interval [-1.1, 1.1]. In addition the state v should be bounded to the interval [-0.5, 1.7].

The state vector is defined by $\mathbf{x} = [s, v, m]^T$.

The objective of the control task is to transite the system from the start state $\mathbf{x}_s = [0, 0, 1]^T$ to the final state $\mathbf{x}_f = [10, 0, -]^T$.

In addition to the other examples, the third state of the final state vector is not fixed. The final mass is not known a-prioi and can be chosen by the optimizer in order to minimze the objective function. But we initialize/estimate the final value to $m_f = 0.5$.

The system dynamics are specified using the FreeSpaceRocketSystem class. See the following header file how to add the nonlinear state space model mentioned above.

File **rocket_system.h:**

```
#ifndef __teb_package__examples_rocket_system__
#define __teb_package__examples_rocket_system__


#include <teb_package.h>


class FreeSpaceRocketSystem : public teb::SystemDynamics<3,1>
{
public:
  using StateVector = teb::SystemDynamics<3,1>::StateVector;
  using ControlVector = teb::SystemDynamics<3,1>::ControlVector;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      StateVector state_equations;

      state_equations[0] = x[1]; // sdot = v
      state_equations[1] = (u[0] - 0.02 * x[1] * x[1]) / x[2]; // (u-0.02*v^2)/m
      state_equations[2] = -0.01 * u[0] * u[0]; // -0.01 * u^2


      return state_equations;
  }

};


#endif /* defined(__teb_package__examples_rocket_system__) */
```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the FreeSpaceRocketSystem object specified in van_der_pol_system.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows (**Note**, the number of iterations and weights of the soft constraints is too low using the default settings, therefore the constraints are not satisfied completely after the first optimization):

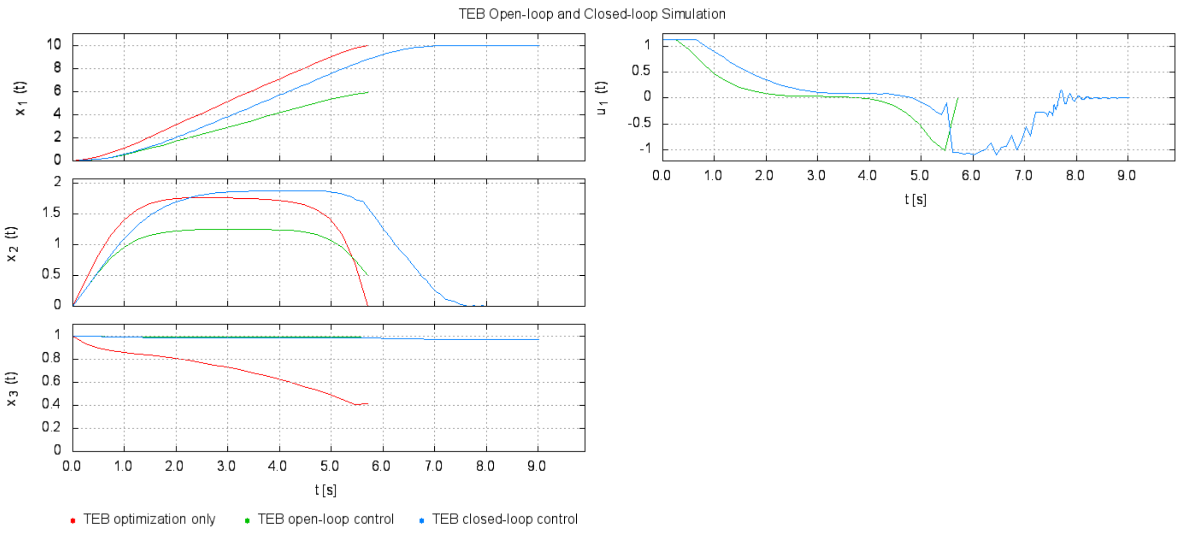


Figure 18.13: Result of rocket_system.cpp

File **rocket_system.cpp**

```
#include "rocket_system.h"

int main()
{
    const int p = 3; // number of states

    // Change default config, since default soft constraint weights are too low.
    teb::Config cfg;
    cfg.optim.solver.lsq.weight_inequalities = 50;
    cfg.optim.solver.lsq.weight_adaptation_factor = 1.1;
    cfg.optim.solver.lsq.soft_constr_epsilon = 0;

    // Define start and goal
    double x0[] = {0,0,1}; // start state
    double xf[] = {10,0,0.5}; // goal state
    bool goal_fixed[] = {true,true,false}; // mass is not fixed

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&cfg, &solver);

    // System specific settings
    FreeSpaceRocketSystem system;

    teb.setSystemDynamics(&system);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.1, 1.1);
    teb.activateStateBounds(1,-0.5,1.7);
    teb.setGoalStatesFixedOrUnfixed(goal_fixed);

    // Create simulator
    teb::TebPlotter plotter;
    teb::Simulator<p,1> sim(&teb,system,&plotter);

    // Set simulation sample time
    sim.setSampleTime(-1); // Inherit from TEB (asynchronous control)

    // Simulate 9 seconds
    sim.simOpenAndClosedLoop(x0,xf,9);


    return 0;
}
```

18.11 van_der_pol_system.cpp

The following example shows the TEB controller applied to the Van der Pol oscillator (http://en.wikipedia.org/wiki/Van_der_Pol_oscillator)

$$\ddot{x}(t) - a(1 - x^2(t))\dot{x}(t) + x(t) = u(t)$$

The control input u should be bounded to the interval [-1, 1].

The state vector is defined by $\mathbf{x} = [x, \dot{x}]^T$. In this example it is $a = 1$.

The objective of the control task is to transite the system from the start state $\mathbf{x}_s = [0,0]^T$ to the final state $\mathbf{x}_f = [1,0]^T$.

The system dynamics are specified using the VanDerPolSystem class. See the following header file how to add the nonlinear state space model mentioned above.

File **van_der_pol_system.h:**

```
#ifndef __teb_package__examples_van_der_pol_system__
#define __teb_package__examples_van_der_pol_system__


#include <teb_package.h>


class VanDerPolSystem : public teb::SystemDynamics<2,1>
{
public:
  using StateVector = teb::SystemDynamics<2,1>::StateVector;
  using ControlVector = teb::SystemDynamics<2,1>::ControlVector;

  inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
      Eigen::Ref<const ControlVector>& u) const
  {
      StateVector state_equations;

      state_equations[0] = x[1]; // xdot = xdot
      state_equations[1] = - _a * (x[0] - 1) * x[1] - x[0] + u[0];


      return state_equations;
  }

    void setParameters(double a) {_a = a;};

protected:
    double _a = 1;

};


#endif /* defined(__teb_package__examples_van_der_pol_system__) */
```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the VanDerPolSystem object specified in van_der_pol_system.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows:

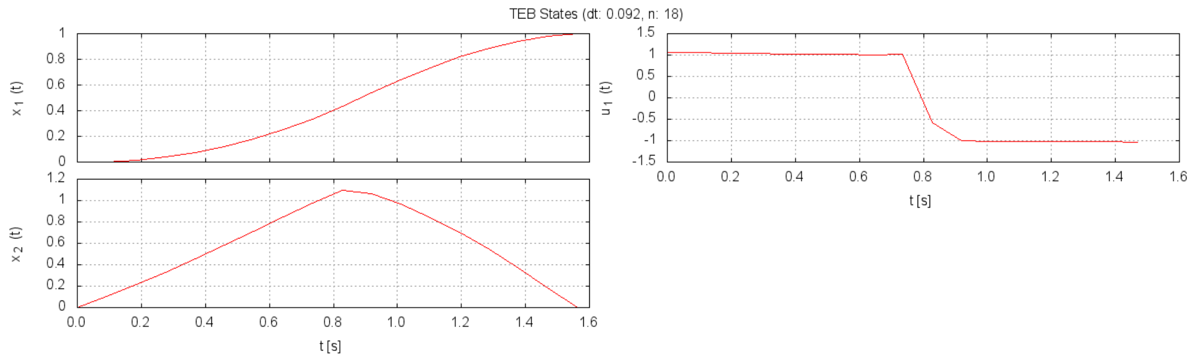


Figure 18.14: Result of van_der_pol_system.cpp

File **van_der_pol_system.cpp**

```
#include "van_der_pol_system.h"


int main()
{

    const int p = 2; // number of states


    // Define start and goal
    double x0[] = {0,0}; // start state
    double xf[] = {1,0}; // goal state
    double u;

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&solver);

    // System specific settings
    VanDerPolSystem system;
    system.setParameters(1.0);

    teb.setSystemDynamics(&system);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);

    // Perform open-loop planning
    START_TIMER;
    teb.step(x0, xf, &u);
    STOP_TIMER("TEB optimization loop");

    // Print determined control for the current sampling interval
    PRINT_INFO("Control u: " << u);

    // Plot states and control input
    teb::TebPlotter plotter;
    plotter.plotTEB(teb);

    return 0;
}
```

Bibliography

[1] Martin Keller, Frank Hoffmann, Torsten Bertram, Carsten Hass, and Alois Seewald. Planning of optimal collision avoidance trajectories with timed elastic bands. In *19th World Congress of the International Federation of Automatic Control*, pages 9822–9827, 2014. 2

[2] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011. 2, 9, 53, 63, 80, 149, 204, 207, 216, 219, 359

[3] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *7th German Conference on Robotics (ROBOTIK)*, pages 74–79, Berlin and Offenbach, 2012. VDE-Verl. 2, 359, 360, 361

[4] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. Efficient trajectory optimization using a sparse model. In *European Conference on Mobile Robots*, pages 138–143, 2013. 2

[5] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Prädiktive regelung mit timed-elastic-bands. *at - Automatisierungstechnik*, 62(10):720–731, 2014. 1, 293

Index