Documentation of ISeeML (Introducing a Smooth, Efficient and Easy-to-use Motion Library)

Alexis Scheuer

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Documentation of ISeeML (Introducing a Smooth, Efficient and Easy-to-use Motion Library)

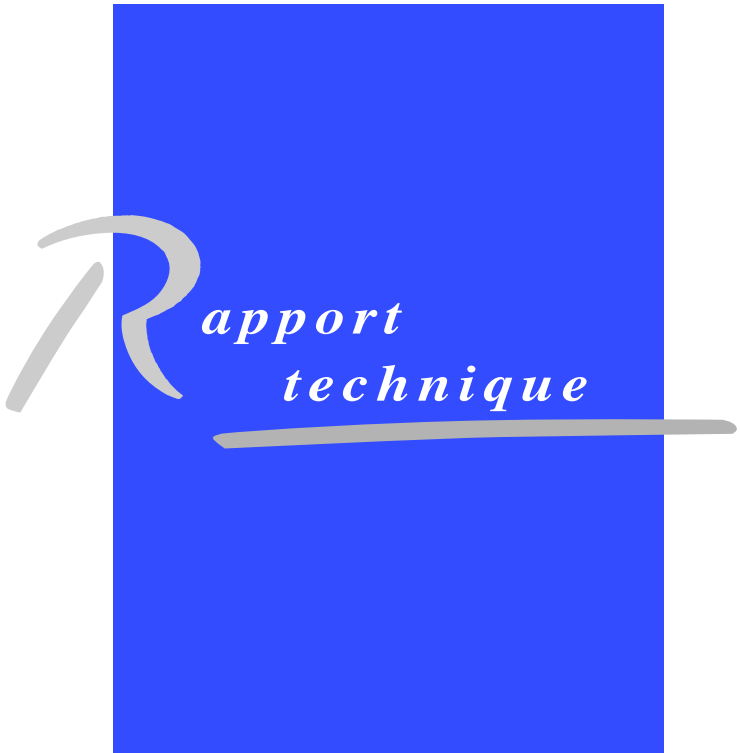
http://iseeml.loria.fr

Alexis Scheuer
CS Dept, Henri Poincaré (Nancy I) University &
MAIA Team, INRIA Nancy – Grand Est / LORIA Lab.
Campus Scientifique — BP 239
54506 Vandœuvre-lès-Nancy cedex FRANCE

http://www.loria.fr/~scheuer – Alexis.Scheuer@loria.fr

N° 0396

September 2010



ISSN 0249-0803 ISRN INRIA/RT--0396--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
NANCY - GRAND EST

Documentation of ISeeML
(Introducing a Smooth, Efficient and Easy-to-use Motion Library)

http://iseeml.loria.fr

Alexis Scheuer
CS Dept, Henri Poincaré (Nancy I) University &
MAIA Team, INRIA Nancy – Grand Est / LORIA Lab.
Campus Scientifique — BP 239
54506 Vandœuvre-lès-Nancy cedex FRANCE
http://www.loria.fr/~scheuer – Alexis.Scheuer@loria.fr

Thème : Représentation et traitement des données et des connaissances

Équipe-Projet Maia

Rapport technique n° 0396 — September 2010 — 127 pages

Abstract: This technical report introduces ISeeML, a C++ library to compute efficient (near to the optimal) motions for mobile robots respecting strong non-holonomic constraints. It contains a short presentation of the library as well as a detailed documentation of the classes offered by this library.

This first published version of the library only handles forward-only path generation for car-like robots, but future versions will propose other types of motions, considering manoeuvres, obstacle avoidance and other kind of robots.

Key-words: Mobile robotics, non-holonomic motion, continuous-curvature path, clothoids, steering method

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Documentation de ISeeML
(Introducing a Smooth, Efficient and Easy-to-use Motion Library)

http://iseeml.loria.fr

Résumé : Ce rapport technique présente ISeeML, une bibliothèque écrite en C++ et qui permet de calculer des mouvements efficaces (proche de l'optimal) pour des robots mobiles respectant de fortes contraintes non holonomes. Il contient une courte présentation de la bibliothèque, ainsi que une documentation détaillée des classes proposées par cette bibliothèque.

Cette première version publiée de la bibliothèque ne permet que la génération de chemins en marche avant pour des robots de type voiture, mais les versions futures proposeront d'autres types de mouvements, prenant en compte les manœuvres, l'évitement d'obstacles et d'autres genres de robots.

Mots-clés : Robotique mobile, mouvement non holonome, chemin à courbure continue, clothoïdes, "steering method"

Contents

Chapter 1

Presentation

1.1 Motivation

The main interest of this library is to offer smooth (continuous-curvature) efficient (close to the optimal) motions for mobile robots [7, 8].

This first version handles forward-only paths: curvature continuity has less importance when the robot can stop, and efficient selection of locally optimal parts still needs complementary studies.

Obtained paths correspond to locally optimal motions with constant velocity for mobile robots, either car-like or with differential-wheels (e.g. Hilare or Khepera). Classical paths (with a discontinuous curvature profile, called Dubins' paths [4, 1, 2]) are also provided. Both paths can also be used for aerial robots, as the motion constraints of those are similar to those of mobile robots.

More details can be found in the articles listed in the bibliography. Smooth motions for other type of robots will be handled in future versions.

1.2 Researchers involved

This library has currently been written by a unique author (help for future versions is welcome):

Alexis Scheuer Alexis Scheuer is « maître de conférence » (assistant professor) at Université Henri Poincaré, Nancy I (`http://www.uhp-nancy.fr`). He is a member of MAIA team (`http://maia.loria.fr`), of LORIA laboratory (`http://www.loria.fr`) and INRIA Nancy – Grand Est (`http://www.inria.fr/nancy`).

However, this library is issued from a research work which has been accomplished thanks to the help of the following persons:

- Thierry Fraichard is researcher in e-Motion team (`http://emotion.inrialpes.fr`), of LIG laboratory (`http://lig.imag.fr`) and INRIA Rhône-Alpes (`http://www.inrialpes.fr`); he supervised A. Scheuer's thesis [7, 8];
- Juan-Manuel Ahuactzin, who is currently research head at Probayes Co. (`www.probayes.com`), collaborates with e-Motion team since 1991; he integrated ISeeML's steering method into his planning scheme called "Ariadne's Clew" [5] (*cf* v2.0 planned improvement);
- Pauline Milhe considered trajectory planning as a generalization of A. Scheuer's work, during a training course for her last year at INSA Toulouse (`http://www.gmm.insa-tlse.fr`);
- Richard Desvigne generalized the results obtained by A. Scheuer for forward-only paths to paths with backup maneuvers, during a training course for his last year at HEI Lille (`http://www.hei.fr`) [6] (*cf* v1.3 planned improvement);
- Guillaume Maillard defined ISeeML's second[1] graphic user interface, based on GTK and Gnome, during two training courses in MAIA team for his second year at the École Supérieure d'Informatique et d'Applications de Lorraine (`http://www.esial.uhp-nancy.fr`);
- Hubert Cecotti defined a hybrid (stochastic/analytic) planning scheme, using ISeeML's steering method together with a Markov Decision Process (MDP) [3] (*cf* v2.0 planned improvement), during his Nancy I « DEA » (Master) training course; he is currently working in the Institute of Automation of the University of Bremen (`http://www.iat.uni-bremen.de`), in Germany.

1.3 Usage

This version of ISeeML only provides forward path generation (no obstacle avoidance, or planning) with or without continuous curvature profile.

ISeeML does not have yet a standard Graphic User Interface. This means you can only, for the moment, compute forward-only paths with continuous or discontinuous curvature profile, and get their characteristics.

A graphical example is however provided, using wxWindows (`http://www.wxWindows.org`). Integrating GUI into the class hierarchy is planned for next version, and back-up manoeuvres (change of direction of motion) for the following one.

[1] ISeeML's first GUI was based on Leda library (`http://www.mpi-sb.mpg.de/LEDA`). Currently, ISeeML's GUI uses wxWidgets (`www.wxWidgets.org`), which seems easier to use and more portable than GTK.

This library installation has no requirement, except of course a C++ compiler. Depending on the one used, you may need to add optional includes or lines. To try the GUI example, you however need to install wxWindows.

This library has been compiled and tested:

- on several computers (Intel Core2 Duo, Xeon, Genuine Intel, Pentium IV, Pentium III, KoreBot II in a Khepera III),
- under several OS (Linux: Ubuntu 10.04-06.06, Red Hat; Windows: 98, XP SP2 & SP3),
- with various compilers (g++: Linux and Cygwin; Borland C++ Builder, Visual Studio).

Computation time varies a lot, depending on:

- the processor (KoreBot II is about 20 times slower than the Pentium III, which is about 14 times slower than the Intel Core2 Duo)
- the OS (computation is at least twice faster under Linux than under Windows),
- the compilers (under Windows,computation is 3 time faster using Visual C++ or Borland C++ than using g++),
- the path computed (Dubins' paths are computed twice faster than continuous-curvature paths, called FSC).

They vary from 2 ms (FSC paths under Linux on KoreBot II) to 5 µs (Dubins' paths under Linux on Intel Core2 Duo). Hundreds of millions tests have been done, with pre- and post-conditions activated (cf. CompilerFlags.h), without any error.

Fwd.cpp, TimeFwd.cpp, LengthFwd.cpp and wxGuiFwd.cpp are four examples showing how classes of this library can be used (the second example gives computation times, the fourth ones is a GUI).

1.4 Improvements Planned

Description	Status
Definition of a portable GUI	At work (v1.1)
More details in the documentation	Planned (v1.2?)
Addition of back-up maneuvres [6]	Planned (v1.2?)
Collision detection	Planned (v1.3?)
Planning amidst obstacles [7]	Planned (v1.3?)
Multiple planning methods amidst obstacles	Planned (v2.0?)
Etc.	Later (v2+?)

Chapter 2

Package Documentation

2.1 Modules Documentation

SCPLS is made of three modules, containing respectively basic classes, 2D geometric classes and robotic classes, and a fourth one giving some demonstration programs (showing how to use the previous classes).

2.1.1 Basic Classes

This module contains basic classes and files, used by other modules.

- File iSeeML/CompilerFlags.h defines relations between compilation flags.
- File iSeeML/NameSpaces.hpp defines ISeeML's namespaces.
- Class iSeeML::Object is inherited by all other classes of ISeeML.

2.1.2 2D Geometric Classes

This module contains geometric classes.

- Class iSeeML::geom::Object is inherited by all other geometric classes of ISeeML. Geometric object can be separated in two sets:
  1. A iSeeML::geom::BasicObject is a basic geometric object, from which more complex objects can be defined.
     (a) A iSeeML::geom::Vector is a 2D vector.
     (b) A iSeeML::geom::Point is a 2D point.

2. Complex geometric objects will be composed of several basic geometric objects. It will be the case of some geometric paths (the geometric projection of robotic compound paths, for example).

2.1.3 Robotic Classes

This module contains robotic classes, which can be (for the moment) configurations (giving the robot's state) or path (giving the robot's motion). Other classes' type will be added in later versions (robots, defining the robot itself, dynamic configurations and trajectories, adding time to geometry in configurations and paths, etc.).

- Class iSeeML::rob::Object is inherited by all other robotic classes of ISeeML.
  - A iSeeML::rob::OrPtConfig is a classical configuration for mobile robot, giving a position and an orientation.
    - A iSeeML::rob::CurvConfig is similar to previous configuration, with the addition of a curvature field for continuous-curvature planning.
  - Class iSeeML::rob::Path is inherited by all path classes of ISeeML. It defines several virtual methods and a comparison method, common to all paths. As for geometric objects, paths can be separated in two sets:
    1. iSeeML::rob::BasicPath defines basic paths, from which more complex paths are made.
       - A iSeeML::rob::LinCurvPath is (in this version) the only basic path. Its curvature derivative (with respect to the arc length) is constant.
    2. A iSeeML::rob::CompoundPath is a compound path, represented by several basic paths.
       - A iSeeML::rob::DubinsLikePath is a path computed using Dubins' method. It is composed of iSeeML::rob::LinCurvPath.
         - A iSeeML::rob::DubinsPath is a Dubins' path, with a discontinuous curvature profile.
         - A iSeeML::rob::FscPath is very similar to a Dubins' path, except that its curvature profile is continuous.

2.1.4 Demonstration programs

This module contains files and classes used to build demonstration programs.

- File Fwd.cpp is the first application defined, and is used to verify whether construction of Dubins' and FSC paths connecting two given configurations are correct.

  This verification is now done by the two following test programs, this one being only given as the simplest use of ISeeML.

- File TimeFwd.cpp is the second application defined. It allows to find Dubins' and FSC paths' (average) computation times, and to verify computations (using CompilerFlags.h).

- File LengthFwd.cpp is very similar to the previous one, except that it computes paths' length instead of computation times.

- File wxGuiFwd.cpp was ISeeML's first Graphical User Interface. It draws Dubins' and FSC paths to connect a fixed configuration to a moving one, whose position follows the mouse and whose orientation can be changed using keyboard. This interface uses wxWidgets (`http://www.wxWidgets.org`), which should become a base for ISeeML v1.1 GUI toolkit.

2.2 Namespace List

The namespaces of ISeeML, as defined in file iSeeML/NameSpaces.hpp, correspond exactly to the modules:

- the including namespace `iSeeML` contains the other namespaces and the classes of the basic module (*cf.* § 2.1.1, p. 5),
- the geometric namespace `iSeeML::geom` contains the classes of the 2D geometric module (*cf.* § 2.1.2, p. 5), and
- the robotic namespace `iSeeML::rob` contains the classes of the robotic module (*cf.* § 2.1.3, p. 6).

2.3 Class Hierarchy

The inheritance list, also similar to the module contents, is:

2.4 Classes and Files Documentation

2.4.1 SCPLS Namespace

2.4.1.1 include/iSeeML/CompilerFlags.h File Reference

Definition of relations between ISeeML compilation flags.

Detailed Description

Definition of relations between ISeeML compilation flags. ISeeML compilation flags are hierarchicaly defined: for example, definition of `ISEEML_CHECKS` activates all ISeeML checking flags. They can be activated by adding in makefile a `-D` option for the compiler. Possible flags are:

- `ISEEML_CHECKS` (checking everything)
  - `ISEEML_CHECK_PRECOND` (checking preconditions)
    - `ISEEML_CHECK_BASIC_PRECOND` (checking preconditions for basic objects)
      - `ISEEML_CHECK_OBJECT_PRECOND` (checking preconditions for ISeeML' objects)
    - `ISEEML_CHECK_GEOM_PRECOND` (checking preconditions for geometric module)
      - `ISEEML_CHECK_GEOM_VECT_PRECOND` (checking preconditions for vectors)
      - `ISEEML_CHECK_GEOM_POINT_PRECOND` (checking preconditions for points)
    - `ISEEML_CHECK_ROB_PRECOND` (checking preconditions for robotic module)
      - `ISEEML_CHECK_CONFIG_PRECOND` (checking preconditions for configurations)
      - `ISEEML_CHECK_ORPT_CONFIG_PRECOND` (checking preconditions for standard configurations)

- ISEEML_CHECK_CURV_CONFIG_PRECOND (checking preconditions for curvature configurations)
- ISEEML_CHECK_PATH_PRECOND (checking preconditions for paths)
- ISEEML_CHECK_LIN_PATH_PRECOND (checking preconditions for linear basic paths)
- ISEEML_CHECK_COMP_PATH_PRECOND (checking preconditions for compound paths)
- ISEEML_CHECK_DLIKE_PATH_PRECOND (checking preconditions for Dubins-like paths)
- ISEEML_CHECK_DUBINS_PATH_PRECOND (checking preconditions for Dubins' paths)
- ISEEML_CHECK_FSC_PATH_PRECOND (checking preconditions for FSC paths)

– ISEEML_CHECK_POSTCOND (checking postconditions)

* ISEEML_CHECK_PATH_POSTCOND (checking postconditions for robotic paths)
  - ISEEML_CHECK_COMP_PATH_POSTCOND (checking postconditions for compound paths)
  - ISEEML_CHECK_DLIKE_PATH_POSTCOND (checking postconditions for Dubins-like paths)
  - ISEEML_CHECK_DUBINS_PATH_POSTCOND (checking postconditions for Dubins' paths)
  - ISEEML_CHECK_FSC_PATH_POSTCOND (checking postconditions for FSC paths)

– ISEEML_CHECK_POINTERS (checking pointer manipulations)

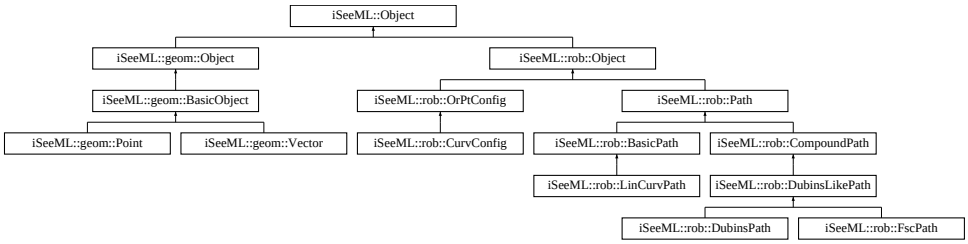
* ISEEML_CHECK_NULL_POINTER (checking for invalid reference)
* ISEEML_CHECK_ARRAY_ELEMT (checking for incorrect array's index)

2.4.1.2 iSeeML::Object Class Reference

This class is the base class of ISeeML: every class of ISeeML inherits this one.

`#include <Object.hpp>`

Inheritance diagram for iSeeML::Object:



Public Member Functions

- virtual ∼Object ()

  This virtual class needs a virtual destructor.

- virtual const string className () const

  Description method, giving the object's class name.

- virtual iSeeML::Object & clone () const =0

  Description method, giving a copy of the current object.

- virtual void writeTo (ostream &O) const =0

  Description method, writing a description of the object into a given output stream.

- bool sameClass (const iSeeML::Object &other)

  Method verifying that a given object has the same type (same class name) than the current one.

Static Public Member Functions

- template<class T >
  static const T & min (const T &a, const T &b)

  Method giving the minimum of two elements.

- template<class T >
  static const T & max (const T &a, const T &b)

  Method giving the maximum of two elements.

- static double sqr (const double &x)

  Method giving the square of a double.

- static double mod2pi (const double &theta)

Method giving the angle between - π (excluded) and π (included), which is equal to a given angle modulo 2 π.

- static bool isPositive (const double &x)

  Method telling whether a double is strictly positive.

- static bool isNegative (const double &x)

  Method telling whether a double is strictly negative.

- static bool isZero (const double &x)

  Method comparing a double to zero.

- static int sign (const double &x)

  Method giving the sign of a double.

Protected Member Functions

- virtual int algDimension () const

  Description method for algebraic vectors, giving the dimension of the containing space (default is zero).

- virtual double algCoord (const int i) const

  Description method for algebraic vectors, giving coordinate of given index for this vector.

- virtual void algWriteTo (ostream &O) const

  Description method for algebraic vectors, writing this vector in a given output stream: coordinates for each dimension are written, between paranthesis and separated by commas.

Related Functions

(Note that these are not member functions.)

- ostream & operator<< (ostream &O, const iSeeML::Object &o)

  Modification method, writing a description of a ISeeML object in a given output stream.

Detailed Description

This class is the base class of ISeeML: every class of ISeeML inherits this one. Every ISeeML object has to overload className(), clone() and writeTo() virtual methods. Method className is used to detect problems with virtual methods (e.g. "equality operator" of geometric objects). Method clone is used in arrays of objects. Method writeTo is used by output stream's operator<<.

This class also provides a set of common methods (sqr, ..., max), some of them using a private static small value.

Author:

Alexis Scheuer.

Version:

1.0

Member Function Documentation

► **virtual double iSeeML::Object::algCoord (const int *i*) const [inline, protected, virtual]** Description method for algebraic vectors, giving coordinate of given index for this vector. This method is not in a seperate class, to avoid multiple inheritance (which drastically increases the computation time).

Parameters:

i the index of the searched coordinate.

Precondition:

an error message is generated if check flag `ISEEML_CHECK_OBJECT_PRECOND` (see CompilerFlags.h) is defined, as this method should not be called (it should be overloaded).

Returns:

0.

Reimplemented in iSeeML::geom::Point, iSeeML::geom::Vector, iSeeML::rob::CurvConfig, and iSeeML::rob::OrPtConfig.

► **virtual int iSeeML::Object::algDimension () const [inline, protected, virtual]** Description method for algebraic vectors, giving the dimension of the containing space (default is zero). This method is not in a seperate class, to avoid multiple inheritance (which drastically increases the computation time).

This representation is mainly used to define a unified writeTo. Dimension cannot be static, as it is virtual.

Reimplemented in iSeeML::geom::Point, iSeeML::geom::Vector, iSeeML::rob::CurvConfig, and iSeeML::rob::OrPtConfig.

► **virtual void iSeeML::Object::algWriteTo (ostream & *O*) const `[inline, protected, virtual]`** Description method for algebraic vectors, writing this vector in a given output stream: coordinates for each dimension are written, between paranthesis and separated by commas. This method is not in a seperate class, to avoid multiple inheritance (which drastically increases the computation time).

Parameters:

O the output stream in which description is written.

See also:

algCoord, algDimension, writeTo.

► **virtual const string iSeeML::Object::className () const `[inline, virtual]`** Description method, giving the object's class name. This method is virtual in order to return the correct value through polymorphism, while a static field is also defined (but this one is private in abstract class).

Class name is used in error messages, and to verify type in redefinitions of some virtual methods.

Returns:

the class name as a string.

See also:

ClassName.

► **virtual iSeeML::Object& iSeeML::Object::clone () const `[pure virtual]`** Description method, giving a copy of the current object. This clone is dynamically allocated (and can be built easily using a copy constructor), it has to be deleted later.

Precondition:

memory should not be full. If memory is full and `ISEEML_CHECK_NIL_POINTER` is defined (see CompilerFlags.h), an error message is generated and the program exits.

Returns:

a copy/clone of the current object.

Implemented in iSeeML::geom::Point, iSeeML::geom::Vector, iSeeML::rob::CurvConfig, iSeeML::rob::DubinsPath, iSeeML::rob::FscPath, iSeeML::rob::LinCurvPath, and iSeeML::rob::OrPtConfig.

► **static bool iSeeML::Object::isNegative (const double & *x*) [inline, static]** Method telling whether a double is strictly negative. Computes the double's opposite, and checks if it is positive.

Parameters:

x the double checked.

Returns:

true if the opposite of the double is positive.

See also:

isPositive.

► **static bool iSeeML::Object::isPositive (const double & *x*) [inline, static]** Method telling whether a double is strictly positive.

Parameters:

x the double checked.

Returns:

true if the double is bigger than ISeeML' small value.

► **static bool iSeeML::Object::isZero (const double & *x*) [inline, static]** Method comparing a double to zero.

Parameters:

x the double compared to zero.

Returns:

true if the double's absolute value is smaller than ISeeML' small value.

Examples:

wxGuiFwd.cpp.

► **template<class T > static const T& iSeeML::Object::max (const T & *a*, const T & *b*)** `[inline, static]` Method giving the maximum of two elements. This method uses templates, the elements do not have to be ISeeML objects.

Parameters:

a A first element,
b a second one.

Returns:

the maximum between the two given elements.

► **template<class T > static const T& iSeeML::Object::min (const T & *a*, const T & *b*)** `[inline, static]` Method giving the minimum of two elements. This method uses templates, the elements do not have to be ISeeML objects.

Parameters:

a A first element,
b a second one.

Returns:

the minimum between the two given elements.

► **static double iSeeML::Object::mod2pi (const double & *theta*)** `[inline, static]` Method giving the angle between - π (excluded) and π (included), which is equal to a given angle modulo 2 π.

Parameters:

theta an angle.

Returns:

the given angle modulo 2 π, between - π (excluded) and π (included).

► **bool iSeeML::Object::sameClass (const iSeeML::Object & *other*)** `[inline]` Method verifying that a given object has the same type (same class name) than the current one.

Parameters:

other the given object.

Returns:

whether the two objects have the same class name.

See also:

className.

► **static int iSeeML::Object::sign (const double & *x*) `[inline, static]`**
Method giving the sign of a double.

Parameters:

x the double which sign is searched.

Returns:

the sign of the double, as an integer in {-1, 0, 1}.

See also:

isZero, isPositive, isNegative.

► **static double iSeeML::Object::sqr (const double & *x*) `[inline, static]`**
Method giving the square of a double.

Parameters:

x the double whose square is wanted.

Returns:

the square of the given double.

► **virtual void iSeeML::Object::writeTo (ostream & *O*) const `[pure virtual]`** Description method, writing a description of the object into a given output stream. This method is preferred to a toString method, as writting into a stream is much easier than writting into a string.

Parameters:

O the output stream in which description is written.

Implemented in iSeeML::geom::Point, iSeeML::geom::Vector, iSeeML::rob::CompoundPath, iSeeML::rob::CurvConfig, iSeeML::rob::DubinsPath, iSeeML::rob::FscPath, iSeeML::rob::LinCurvPath, and iSeeML::rob::OrPtConfig.

Friends And Related Function Documentation

► **ostream & operator<< (ostream & *O*, const iSeeML::Object & *o*) `[related]`** Modification method, writing a description of a ISeeML object in a given output stream.

Parameters:

O the output stream in which description is written,
o the ISeeML object whose description is written.

Returns:

the output stream, after writing in it.

See also:

iSeeML::Object::writeTo.

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

- include/iSeeML/Object.hpp

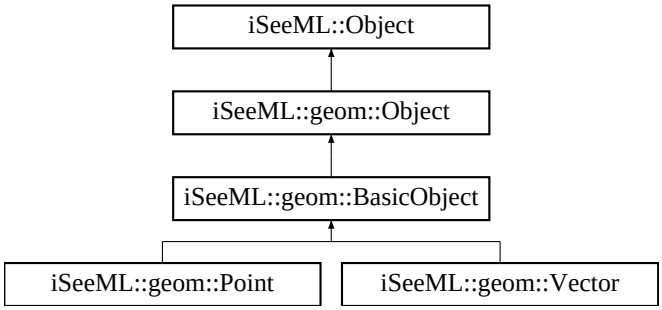
2.4.2 Geometric Namespace

2.4.2.1 iSeeML::geom::Object Class Reference

This class is the base class of the 2D geometric namespace of ISeeML: every geometric object inherits this one.

`#include <Object.hpp>`

Inheritance diagram for iSeeML::geom::Object:



Public Member Functions

- virtual iSeeML::geom::Object & translate (const iSeeML::geom::Vector &v)=0

  Modification method, translating a iSeeML::geom::Object along a vector.

- virtual bool operator== (const iSeeML::geom::Object &other) const

  Equality operator between geometric objects.

Detailed Description

This class is the base class of the 2D geometric namespace of ISeeML: every geometric object inherits this one. Every geometric object has to overload translate method and operator==(const iSeeML::geom::Object&) const.

Author:

Alexis Scheuer.

Version:

1.0

Member Function Documentation

► **virtual bool iSeeML::geom::Object::operator== (const iSeeML::geom::Object & *other*) const** `[inline, virtual]` Equality operator between geometric objects. This cannot be a pure virtual method, as type of the parameter will change. This method should however always be overloaded, and therefore returns false.

Parameters:

other another geometric object.

Returns:

false (this method should be overloaded).

► **virtual iSeeML::geom::Object& iSeeML::geom::Object::translate (const iSeeML::geom::Vector & *v*)** `[pure virtual]` Modification method, translating a iSeeML::geom::Object along a vector.

Parameters:

v the vector of the translation.

Returns:

the object once modified.

Implemented in iSeeML::geom::Point, and iSeeML::geom::Vector.

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

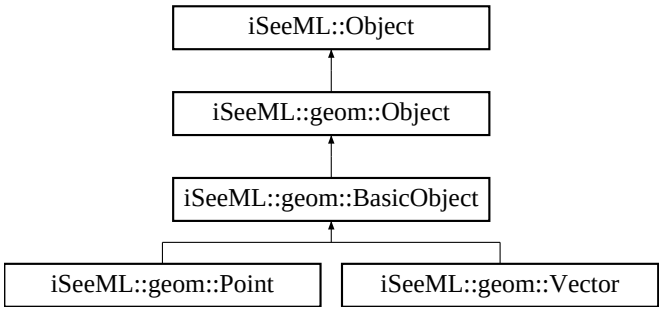
- include/iSeeML/geom/Object.hpp

2.4.2.2 iSeeML::geom::BasicObject Class Reference

This class defines basic 2D geometric objects, from which compound geometric objects are built.

`#include <BasicObject.hpp>`

Inheritance diagram for iSeeML::geom::BasicObject:



Detailed Description

This class defines basic 2D geometric objects, from which compound geometric objects are built.

Author:

Alexis Scheuer.

Version:

1.0

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

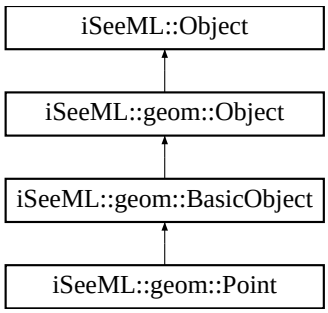
- include/iSeeML/geom/BasicObject.hpp

2.4.2.3 iSeeML::geom::Point Class Reference

This class defines 2D geometric points, defined by their Cartesian coordinates.

`#include <Point.hpp>`

Inheritance diagram for iSeeML::geom::Point:



Public Member Functions

- Point ()

  The default constructor, returning the frame origin (0,0).

- Point (const double &x, const double &y)

  The main constructor, creating a point from its Cartesian coordinates (Polar coordinates are not used for points in ISeeML).

- iSeeML::Object & clone () const

  Description method, giving a copy of the current point.

- void writeTo (ostream &O) const

  Description method, writing a description of the point into a stream: Cartesian coordinate in each dimension is written, between paranthesis and separated by commas.

- const double & xCoord () const

  Description method, giving the point's first coordinate.

- const double & yCoord () const

  Description method, giving the point's second coordinate.

- Point & moveTo (const double &x, const double &y)

  Modification method, moving the point to a given position.

- Point & translate (const iSeeML::geom::Vector &v)

  Modification method, translating the point along a vector.

- bool operator== (const Point &other) const

  Equality operator between points.

- Point operator+ (const iSeeML::geom::Vector &v) const

  Sum operator between a point and a vector, giving the translation of the current point along the vector.

- Point operator- (const iSeeML::geom::Vector &v) const

  Difference operator between a point and a vector, giving the translation of the current point along the opposite of the vector.

- iSeeML::geom::Vector operator- (const Point &other) const

  Difference operator between two points, giving the vector connecting these points.

- double distance (const Point &other) const

  Various method, giving the distance between two points.

Static Public Attributes

- static const string ClassName

  The class name is public, as this class can be instanced.

Protected Member Functions

- int algDimension () const

  Description method, giving the dimension of the containing space (2) when this point is considered as an algebraic vector.

- double algCoord (const int i) const

  Description method, giving Cartesian coordinate of given index of the point.

Detailed Description

This class defines 2D geometric points, defined by their Cartesian coordinates. This class contains description methods giving Cartesian coordinates as well as the writeTo

method, a modification method giving translation of the point, the corresponding operator as well as the equality operator and difference operators between a point and a vector or between two points,and at last a method computing the distance separating two points.

Author:

Alexis Scheuer.

Version:

1.0

Examples:

wxGuiFwd.cpp.

Constructor & Destructor Documentation

► **iSeeML::geom::Point::Point (const double & *x*, const double & *y*) `[inline]`** The main constructor, creating a point from its Cartesian coordinates (Polar coordinates are not used for points in ISeeML).

Parameters:

x the first Cartesian coordinate of the point,
y the second Cartesian coordinate of the point.

See also:

xCoord, yCoord.

Member Function Documentation

► **double iSeeML::geom::Point::algCoord (const int *i*) const `[inline, protected, virtual]`** Description method, giving Cartesian coordinate of given index of the point.

Parameters:

i the index of the searched Cartesian coordinate.

Precondition:

the parameter is an index, and should be equal to one or two. If not, and if `ISEEML_CHECK_GEOM_POINT_PRECOND` is defined (see CompilerFlags.h), an error is generated.

Returns:

the double coordinate associated to the index if this one is between 1 and dimension, or 0.

See also:

xCoord, yCoord, algDimension.

Reimplemented from iSeeML::Object.

► **int iSeeML::geom::Point::algDimension () const** `[inline, protected, virtual]` Description method, giving the dimension of the containing space (2) when this point is considered as an algebraic vector. This representation is mainly used to define a unified writeTo. Dimension cannot be static, as it is virtual.

Reimplemented from iSeeML::Object.

► **iSeeML::Object& iSeeML::geom::Point::clone () const** `[inline, virtual]` Description method, giving a copy of the current point. This clone is dynamically allocated (using default copy constructor), it has to be deleted later.

Returns:

a copy/clone of the current point.

Implements iSeeML::Object.

► **double iSeeML::geom::Point::distance (const Point &** ***other*****) const** `[inline]` Various method, giving the distance between two points. It simply computes the length of the connecting vector.

Parameters:

other another point.

Returns:

the distance between the two points.

See also:

operator-(const iSeeML::geom::Point&), iSeeML::geom::Vector::length.

► **Point& iSeeML::geom::Point::moveTo (const double & *x*, const double & *y*) `[inline]`** Modification method, moving the point to a given position. The original point is changed.

Parameters:

x the first coordinate of the position,
y the second coordinate of the position.

Returns:

the point, after tranformation.

See also:

xCoord, yCoord.

► **Point iSeeML::geom::Point::operator+ (const iSeeML::geom::Vector & *v*) const `[inline]`** Sum operator between a point and a vector, giving the translation of the current point along the vector.

Parameters:

v the vector of the translation.

Returns:

the translated point.

See also:

translate.

► **iSeeML::geom::Vector iSeeML::geom::Point::operator- (const Point & *other*) const `[inline]`** Difference operator between two points, giving the vector connecting these points.

Parameters:

other another point.

Returns:

the vector going from the second given point to the first one (the current).

► **Point iSeeML::geom::Point::operator- (const iSeeML::geom::Vector & *v*) const** **[inline]** Difference operator between a point and a vector, giving the translation of the current point along the opposite of the vector.

Parameters:

v the opposite vector of the translation.

Returns:

the translated point.

See also:

operator+, iSeeML::geom::Vector::operator-().

► **bool iSeeML::geom::Point::operator== (const Point & *other*) const** **[inline]** Equality operator between points.

Parameters:

other another point.

Returns:

whether the two points are equal.

See also:

xCoord, yCoord, iSeeML::Object::isZero.

► **Point& iSeeML::geom::Point::translate (const iSeeML::geom::Vector & *v*)** **[inline, virtual]** Modification method, translating the point along a vector. The original point is changed (addition without modification is obtained using operator+(const iSeeML::geom::Vector&) const).

Parameters:

v the vector of the translation.

Returns:

the point, after translation.

Implements iSeeML::geom::Object.

► **void iSeeML::geom::Point::writeTo (ostream & *O*) const `[inline, virtual]`** Description method, writing a description of the point into a stream: Cartesian coordinate in each dimension is written, between paranthesis and separated by commas.

Parameters:

O the output stream in which description is written.

Implements iSeeML::Object.

► **const double& iSeeML::geom::Point::xCoord () const `[inline]`** Description method, giving the point's first coordinate.

Returns:

the point's first coordinate.

Examples:

wxGuiFwd.cpp.

► **const double& iSeeML::geom::Point::yCoord () const `[inline]`** Description method, giving the point's second coordinate.

Returns:

the point's second coordinate.

Examples:

wxGuiFwd.cpp.

Member Data Documentation

► **const string iSeeML::geom::Point::ClassName `[static]`** The class name is public, as this class can be instanced.

See also:

className.

Reimplemented from iSeeML::geom::BasicObject.

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

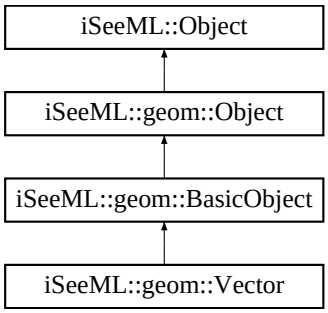
- include/iSeeML/geom/Point.hpp

2.4.2.4 iSeeML::geom::Vector Class Reference

This class defines 2D geometric vectors, which can be defined by their Polar or Cartesian coordinates.

`#include <Vector.hpp>`

Inheritance diagram for iSeeML::geom::Vector:



Public Member Functions

- Vector ()

  The default constructor, creating the zero vector with Cartesian coordinates (0,0).

- Vector (const double &theta)

  A simple constructor, creating the unit vector of given orientation.

- Vector (const double &x, const double &y)

  The main constructor, creating a vector from its Cartesian coordinates.

- iSeeML::Object & clone () const

  Description method, giving a copy of the current vector.

- void writeTo (ostream &O) const

  Description method, writing a description of the vector into a stream: Cartesian coordinate in each dimension is written, between paranthesis and separated by commas.

- const double & xCoord () const

  Description method, giving the vector's first Cartesian coordinate.

- const double & yCoord () const

  Description method, giving the vector's second Cartesian coordinate.

- double orientation () const

  Description method, giving the vector's orientation.

- double length () const

  Description method, giving the vector's length.

- double sqrLength () const

  Description method, giving the square of the vector's length.

- Vector & moveTo (const double &x, const double &y)

  Modification method, moving the point to a given position.

- Vector & translate (const Vector &v)

  Modification method, translating the vector along an other.

- Vector & rotate (const double &theta)

  Modification method, rotating the vector of a given angle.

- Vector & add (const Vector &v)

  Modification method, adding a vector to the current one.

- Vector & multiply (const double &f)

  Modification method, multiplying a vector by a real factor.

- Vector & divide (const double &f)

  Division method, dividing a vector by a real factor.

- Vector & symmetryOx ()

  Modification method, transforming a vector into its symmetric with respect to the X axis.

- Vector & symmetryOy ()

  Modification method, transforming a vector into its symmetric with respect to the Y axis.

- bool operator== (const Vector &other) const

  Equality operator between vectors (differences between both Cartesian coordinates should be zero).

- Vector operator+ (const Vector &v) const

  Sum operator between two vectors, giving the translation of the first one by the other.

- Vector operator- (const Vector &v) const

  Difference operator between two vectors, giving the sum of the first vector and of the second one's opposite.

- Vector operator- () const

  Opposite operator for a vector, such that the sum of the vector and of its opposite is the zero vector.

- Vector operator∗ (const double &f) const

  Multiplication operator between a vector and a real.

- Vector operator/ (const double &f) const

  Division operator between a vector and a real.

- double operator∗ (const Vector &v) const

  Scalar product operator between two vectors.

- double operator^ (const Vector &v) const

  Vectorial product operator between two vectors.

Static Public Member Functions

- static double FresnelCos (const double &s)

  Various method, computing the Fresnel Cosine integral.

- static double FresnelSin (const double &s)

  Various method, computing the Fresnel Sine integral.

- static Vector FresnelInt (const double &s)

  Various method, computing the Fresnel integrals in a vector.

Static Public Attributes

- static const string ClassName

  The class name is public, as this class can be instanced.

Protected Member Functions

- int algDimension () const

  Description method, giving the dimension of the containing space (2) when this vector is considered as an algebraic one.

- double algCoord (const int i) const

  Description method, giving Cartesian coordinate of given index for the vector.

Friends

- Vector operator∗ (const double &f, const Vector &v)

  Multiplication operator between a real and a vector.

- Vector operator/ (const double &f, const Vector &v)

  Division operator between a real and a vector.

Detailed Description

This class defines 2D geometric vectors, which can be defined by their Polar or Cartesian coordinates. This class contains description methods giving Polar and Cartesian coordinates as well as the writeTo method, modification methods giving translation, rotation, scaling and symmetry of a vector, the corresponding operators as well as the equality operator, and a last method computing Cartesian coordinates of a reference clothoid (which are Fresnel integrals).

Author:

Alexis Scheuer.

Version:

1.0

Examples:

wxGuiFwd.cpp.

Constructor & Destructor Documentation

► **iSeeML::geom::Vector::Vector (const double & *theta*) `[inline]`** A simple constructor, creating the unit vector of given orientation. The vector of Polar coordinates `(rho, theta)` can thus be obtained using `iSeeML::geom::Vector(theta).multiply(rho)`.

Parameters:

theta the orientation of the vector.

See also:

orientation, length, multiply.

► **iSeeML::geom::Vector::Vector (const double & *x*, const double & *y*) `[inline]`** The main constructor, creating a vector from its Cartesian coordinates.

Parameters:

x the first Cartesian coordinate of the vector,
y the second Cartesian coordinate of the vector.

See also:

xCoord, yCoord.

Member Function Documentation

► **Vector& iSeeML::geom::Vector::add (const Vector & *v*) `[inline]`** Modification method, adding a vector to the current one. The original vector is changed, addition without modification is obtained using operator+(const Vector&) const.

Parameters:

v the vector of the translation.

Returns:

the new vector, after translation / addition.

► **double iSeeML::geom::Vector::algCoord (const int *i*) const `[inline, protected, virtual]`** Description method, giving Cartesian coordinate of given index for the vector.

Parameters:

i the index of the searched Cartesian coordinate.

Precondition:

the parameter is an index, and should be equal to one or two. If not, and if `ISEEML_CHECK_GEOM_VECT_PRECOND` is defined (see CompilerFlags.h), an error is generated.

Returns:

the Cartesian coordinate associated to the index if this one is between 1 and dimension, or 0.

See also:

xCoord, yCoord, algDimension.

Reimplemented from iSeeML::Object.

► **int iSeeML::geom::Vector::algDimension () const** `[inline, protected, virtual]` Description method, giving the dimension of the containing space (2) when this vector is considered as an algebraic one. This representation is mainly used to define a unified writeTo. Dimension cannot be static, as it is virtual.

Reimplemented from iSeeML::Object.

► **iSeeML::Object& iSeeML::geom::Vector::clone () const** `[inline, virtual]` Description method, giving a copy of the current vector. This clone is dynamically allocated (using default copy constructor), it has to be deleted later.

Returns:

a copy/clone of the current vector.

Implements iSeeML::Object.

► **Vector& iSeeML::geom::Vector::divide (const double &*f*)** `[inline]` Division method, dividing a vector by a real factor. The original vector is changed, division without modification is obtained using operator/() const.

Parameters:

f a real factor.

Precondition:

the dividing factor should not be zero. If it is, and `ISEEML_CHECK_GEOM_VECT_PRECOND` is defined (see CompilerFlags.h), an error is generated.

Returns:

the new vector, after division.

See also:

multiply.

► **static double iSeeML::geom::Vector::FresnelCos (const double & *s*) [inline, static]** Various method, computing the Fresnel Cosine integral.

Parameters:

s the integration length.

Precondition:

Coordinates have only been computed for values of the arc length `s` smaller (in absolute value) then 2, which correspond to a deflection (change of the orientation) of $\pm$ 2 π.

Returns:

the integral, from zero to `s`, of cos(π u^2 / 2).

See also:

FresnelInt, xCoord.

► **static Vector iSeeML::geom::Vector::FresnelInt (const double & *s*) [static]** Various method, computing the Fresnel integrals in a vector. The returned vector correspond to the Cartesian coordinates of the point of given arc length along a reference clothoid (along which curvature is equal to π times the arc length). These coordinates have been precomputed with a precision of 1E-5, and are stored in two static arrays.

Parameters:

s the arc length along the reference clothoid of the searched position, or the integration length of the Fresnel integrals.

Precondition:

Coordinates have only been computed for values of the arc length `s` smaller (in absolute value) then 2, which correspond to a deflection (change of the orientation) of $\pm$ 2 π. An error message is generated if `s` is not correct and check flags `ISEEML_CHECK_GEOM_VECT_PRECOND` or `ISEEML_CHECK_ARRAY_ELEMT` (see CompilerFlags.h) are defined.

Returns:

the vector whose Cartesian coordinates are the integral, from zero to `s`, of respectively cos(π u^2 / 2) and sin(π u^2 / 2).

► **static double iSeeML::geom::Vector::FresnelSin (const double & *s*) [inline, static]** Various method, computing the Fresnel Sine integral.

Parameters:

s the integration length.

Precondition:

Coordinates have only been computed for values of the arc length `s` smaller (in absolute value) then 2, which correspond to a deflection (change of the orientation) of $\pm$ 2 π.

Returns:

the integral, from zero to `s`, of sin(π u^2 / 2).

See also:

FresnelInt, yCoord.

► **double iSeeML::geom::Vector::length () const [inline]** Description method, giving the vector's length. The couple (length, orientation) gives Polar coordinates of the vector, while the couple (xCoord, yCoord) gives Cartesian coordinates.

Returns:

the vector's length.

See also:

sqrLength.

► **Vector& iSeeML::geom::Vector::moveTo (const double & *x*, const double & *y*) [inline]** Modification method, moving the point to a given position. The original vector is changed.

Parameters:

x the first coordinate of the new position,
y the second coordinate of the new position.

Returns:

the vector, after tranformation.

See also:

xCoord, yCoord.

► **Vector& iSeeML::geom::Vector::multiply (const double & *f*) [inline]**
Modification method, multiplying a vector by a real factor. The original vector is changed, multiplication without modification is obtained using operator∗() const.

Parameters:

f a real factor.

Returns:

the new vector, after multiplication.

► **double iSeeML::geom::Vector::operator∗ (const Vector & *v*) const [inline]**
Scalar product operator between two vectors.

Parameters:

v the second vector of the product.

Returns:

the scalar product between the current vector and the given one.

► **Vector iSeeML::geom::Vector::operator∗ (const double & *f*) const [inline]**
Multiplication operator between a vector and a real.

Parameters:

f a real factor.

Returns:

the product of the current vector by the real factor.

See also:

multiply.

► **Vector iSeeML::geom::Vector::operator+ (const Vector & *v*) const [inline]**
Sum operator between two vectors, giving the translation of the first one by the other.

Parameters:

v the vector of the translation.

Returns:

the sum of the two vectors.

See also:

add.

► **Vector iSeeML::geom::Vector::operator- () const** `[inline]` Opposite operator for a vector, such that the sum of the vector and of its opposite is the zero vector.

Returns:

the opposite of the current vector.

See also:

multiply.

► **Vector iSeeML::geom::Vector::operator- (const Vector & *v*) const** `[inline]` Difference operator between two vectors, giving the sum of the first vector and of the second one's opposite.

Parameters:

v a vector.

Returns:

the current vector minus the given one.

See also:

operator-(), operator+.

► **Vector iSeeML::geom::Vector::operator/ (const double & *f*) const** `[inline]` Division operator between a vector and a real.

Parameters:

f a real factor.

Precondition:

the dividing factor should not be zero.

Returns:

the division of the current vector by the real factor.

See also:

operator∗(const double&), divide.

► **bool iSeeML::geom::Vector::operator== (const Vector & *other*) const** `[inline]` Equality operator between vectors (differences between both Cartesian coordinates should be zero).

Parameters:

other another vector.

Returns:

whether the two vectors are equal.

See also:

xCoord, yCoord, iSeeML::Object::isZero.

► **double iSeeML::geom::Vector::operator^ (const Vector & *v*) const** `[inline]` Vectorial product operator between two vectors.

Parameters:

v the second vector of the product.

Returns:

the vectorial product between the current vector and the given one.

► **double iSeeML::geom::Vector::orientation () const** `[inline]` Description method, giving the vector's orientation. The couple (length, orientation) gives Polar coordinates of the vector, while the couple (xCoord, yCoord) gives Cartesian coordinates. Orientation is always uniquely defined, except for zero vector. In that case, we choose to return 0.

Returns:

the vector's orientation, between - π (excluded) and π (included), if vector is not zero, or zero.

See also:

iSeeML::Object::isPositive, iSeeML::Object::isNegative, iSeeML::Object::sign, atan.

► **Vector& iSeeML::geom::Vector::rotate (const double & *theta*) `[inline]`** Modification method, rotating the vector of a given angle. The original vector is changed.

Parameters:

theta the angle of the rotation.

Returns:

the new vector, after rotation.

► **double iSeeML::geom::Vector::sqrLength () const `[inline]`** Description method, giving the square of the vector's length. Uses the scalar product.

Returns:

the square of the vector's length.

See also:

operator∗(const iSeeML::geom::Vector&).

► **Vector& iSeeML::geom::Vector::symmetryOx () `[inline]`** Modification method, transforming a vector into its symmetric with respect to the X axis.

Returns:

the new vector, after transformation.

► **Vector& iSeeML::geom::Vector::symmetryOy () `[inline]`** Modification method, transforming a vector into its symmetric with respect to the Y axis.

Returns:

the new vector, after transformation.

► **Vector& iSeeML::geom::Vector::translate (const Vector & *v*) `[inline, virtual]`** Modification method, translating the vector along an other. The original vector is changed, addition without modification is obtained using operator+(const Vector&) const.

Parameters:

v the vector of the translation.

Returns:

the new vector, after translation.

See also:

add.

Implements iSeeML::geom::Object.

► **void iSeeML::geom::Vector::writeTo (ostream & *O*) const `[inline, virtual]`** Description method, writing a description of the vector into a stream: Cartesian coordinate in each dimension is written, between paranthesis and separated by commas.

Parameters:

O the output stream in which description is written.

Implements iSeeML::Object.

► **const double& iSeeML::geom::Vector::xCoord () const `[inline]`** Description method, giving the vector's first Cartesian coordinate.

Returns:

the vector's first Cartesian coordinate.

► **const double& iSeeML::geom::Vector::yCoord () const `[inline]`** Description method, giving the vector's second Cartesian coordinate.

Returns:

the vector's second Cartesian coordinate.

Friends And Related Function Documentation

► **Vector operator∗ (const double & *f*, const Vector & *v*)** **[friend]** Multiplication operator between a real and a vector.

Parameters:

f the real factor,
v the vector.

Returns:

the product between the factor and the vector.

See also:

const operator∗(const double&).

► **Vector operator/ (const double & *f*, const Vector & *v*)** **[friend]** Division operator between a real and a vector.

Parameters:

f the real factor,
v the vector.

Precondition:

the dividing factor should not be zero.

Returns:

the division between the factor and the vector.

See also:

const operator/(const double&).

Member Data Documentation

► **const string iSeeML::geom::Vector::ClassName** **[static]** The class name is public, as this class can be instanced.

See also:

className.

Reimplemented from iSeeML::geom::BasicObject.

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

- include/iSeeML/geom/Vector.hpp

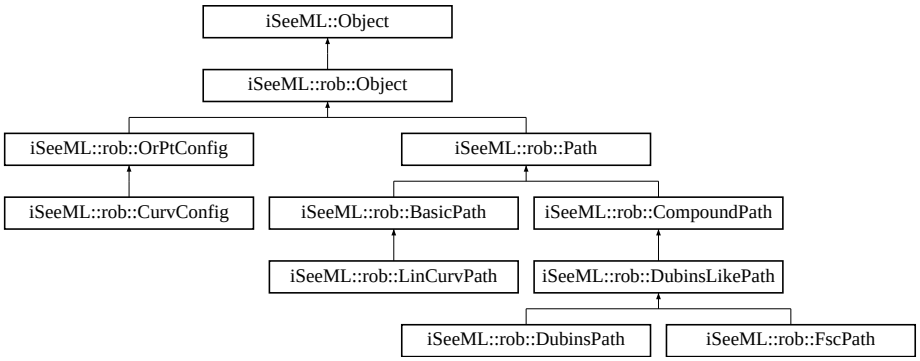
2.4.3 Robotic Namespace

2.4.3.1 iSeeML::rob::Object Class Reference

This class is the base class of robotic namespace of ISeeML: every robotic object inherits this one.

`#include <Object.hpp>`

Inheritance diagram for iSeeML::rob::Object:



Detailed Description

This class is the base class of robotic namespace of ISeeML: every robotic object inherits this one.

Author:

Alexis Scheuer.

Version:

1.0

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

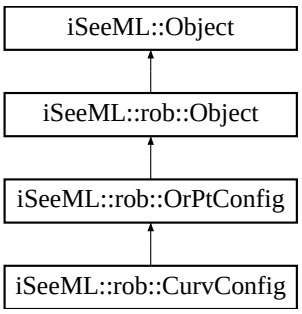
- include/iSeeML/rob/Object.hpp

2.4.3.2 iSeeML::rob::OrPtConfig Class Reference

This class defines a standard robotic configuration, containing the position of a reference point and the orientation of a main axis.

`#include <OrPtConfig.hpp>`

Inheritance diagram for iSeeML::rob::OrPtConfig:



Public Member Functions

- OrPtConfig ()

  The default constructor should only be used for array initializations: it correspond to a default point with zero orientation.

- OrPtConfig (const iSeeML::geom::Point &P, const double &theta)

  The main constructor.

- OrPtConfig (const double &x, const double &y, const double &theta)

  A usefull constructor.

- virtual ∼OrPtConfig ()

  The virtual destructor does nothing.

- iSeeML::Object & clone () const

  Description method, giving a copy of the current configuration.

- void writeTo (ostream &O) const

  Description method, writing the configuration in a given output stream: coordinate in each dimension is written, between paranthesis and separated by commas.

- const iSeeML::geom::Point & position () const

  Description method, giving the position of the robot's reference point in this configuration.

- const double & orientation () const

Description method, giving the orientation of the robot's main axis in this configuration.

- OrPtConfig & uTurn ()

  Modification method, turning the current oriented point to its opposite: the position does not change but the orientation is replaced by its opposite ($\pm \pi$ is added to it).

- bool operator== (const OrPtConfig &other) const

  Equality operator between oriented points.

- OrPtConfig opposite () const

  Various method, giving the opposite oriented point of the current one: the opposite oriented point has the same position but opposite orientation (current oriented point's orientation $\pm \pi$).

- double distance2 (const OrPtConfig &other) const

  Various method, giving the distance between two oriented points' positions.

- bool isParallelTo (const OrPtConfig &other) const

  Various method, checking whether two oriented points are parallel, ie whether their orientations are equal (modulo 2 pi).

- bool isSymmetricTo (const OrPtConfig &other) const

  Various method, checking whether two oriented points are symmetric, ie whether their orientations are symmetric wrt the line segment connecting their position.

- bool isAlignedWith (const OrPtConfig &other) const

  Various method, checking whether two oriented points are aligned: their orientations should be equal, and be the same as the orientation of the segment connecting their positions.

- bool hasInFront (const iSeeML::geom::Point &point) const

  Various method, checking whether a point is in the front half-plane of the current oriented point: the vector connecting the current oriented point's position to the given point should make an angle between - π/ 2 and π/ 2 with the current oriented point's orientation.

Static Public Attributes

- static const string ClassName

  The class name is public, as this class can be instanced.

Protected Member Functions

- int algDimension () const

  Description method, giving the dimension of the containing space (3) when this configuration is considered as an algebraic vector.

- double algCoord (const int i) const

  Description method, giving the coordinate of given index of the configuration (both coordinates of the position, then orientation).

Detailed Description

This class defines a standard robotic configuration, containing the position of a reference point and the orientation of a main axis. This configuration is sufficient to define the position of most robots, and thus is enough for collision avoidance. However, taking into account dynamic constraints (as velocity and acceleration bounds) or more precise kinematic constraints (as continuity of the directing wheels' positions) generally requires a more complex configuration, as e.g. iSeeML::rob::CurvConfig.

This class contains description methods giving position and orientation of a standard configuration as well as the writeTo method, a modification method changing a standard configuration into its opposite, the equality operator, and various methods giving the opposite of a standard configuration, the distance between two standard configurations' positions or whether two such configurations verify some properties (parallelism, symmetry, alignment) or whether a point is in front of a standard configuration.

Author:

Alexis Scheuer.

Version:

1.0

Examples:

Fwd.cpp, LengthFwd.cpp, TimeFwd.cpp, and wxGuiFwd.cpp.

Constructor & Destructor Documentation

► **iSeeML::rob::OrPtConfig::OrPtConfig** () **`[inline]`** The default constructor should only be used for array initializations: it correspond to a default point with zero orientation.

See also:

default constructor of iSeeML::geom::Point.

► **iSeeML::rob::OrPtConfig::OrPtConfig (const iSeeML::geom::Point & *P*, const double & *theta*)** **`[inline]`** The main constructor.

Parameters:

P the position of the reference point,
theta the orientation of the main axis, taken between - π (excluded) and π (included) modulo 2 π.

See also:

iSeeML::Object::mod2pi.

► **iSeeML::rob::OrPtConfig::OrPtConfig (const double & *x*, const double & *y*, const double & *theta*)** **`[inline]`** A usefull constructor.

Parameters:

x the first coordinate of the reference point,
y the second coordinate of the reference point,
theta the orientation of the main axis, taken between - π (excluded) and π (included) modulo 2 π.

See also:

main constructor of iSeeML::geom::Point, iSeeML::Object::mod2pi.

Member Function Documentation

► **double iSeeML::rob::OrPtConfig::algCoord (const int *i*) const** **`[inline, protected, virtual]`** Description method, giving the coordinate of given index of the configuration (both coordinates of the position, then orientation).

Parameters:

i the index of the searched coordinate.

Precondition:

the parameter is an index, and should be equal to one, two or three. If not, and `ISEEML_CHECK_ORPT_CONFIG_PRECOND` is defined (see CompilerFlags.h), an error is generated.

Returns:

the double coordinate associated to the index if this one is between 1 and dimension, or 0.

See also:

orientation, position, iSeeML::geom::Point::xCoord, iSeeML::geom::Point::yCoord.

Reimplemented from iSeeML::Object.

Reimplemented in iSeeML::rob::CurvConfig.

► **int iSeeML::rob::OrPtConfig::algDimension** () **const** `[inline, protected, virtual]` Description method, giving the dimension of the containing space (3) when this configuration is considered as an algebraic vector. This representation is mainly used to define a unified writeTo. Dimension cannot be static, as it is virtual.

Reimplemented from iSeeML::Object.

Reimplemented in iSeeML::rob::CurvConfig.

► **iSeeML::Object& iSeeML::rob::OrPtConfig::clone** () **const** `[inline, virtual]` Description method, giving a copy of the current configuration. This clone is dynamically allocated (using default copy constructor), it has to be deleted later.

Returns:

a copy/clone of the current configuration.

Implements iSeeML::Object.

Reimplemented in iSeeML::rob::CurvConfig.

► **double iSeeML::rob::OrPtConfig::distance2 (const OrPtConfig &** ***other*****) const** `[inline]` Various method, giving the distance between two oriented points' positions. **Note :** this is the standard 2-dimensional plane distance, instead of being a distance in the 3-dimensional space of the oriented points. Its name comes from this fact.

Parameters:

other another oriented point.

Returns:

the distance between the oriented points' positions.

See also:

position, iSeeML::geom::Point::distance.

► **bool iSeeML::rob::OrPtConfig::hasInFront (const iSeeML::geom::Point & *point*) const** `[inline]` Various method, checking whether a point is in the front half-plane of the current oriented point: the vector connecting the current oriented point's position to the given point should make an angle between - π/ 2 and π/ 2 with the current oriented point's orientation. The vector going from the oriented point's position to the point and the vector of same orientation as the oriented point should have a positive scalar product.

Parameters:

point a point.

Returns:

whether the given point is in the front half-plane of the current oriented point.

See also:

position, orientation, iSeeML::geom::Point::operator-, iSeeML::geom::Vector::operator∗(const Vector&).

► **bool iSeeML::rob::OrPtConfig::isAlignedWith (const OrPtConfig & *other*) const** `[inline]` Various method, checking whether two oriented points are aligned: their orientations should be equal, and be the same as the orientation of the segment connecting their positions. In fact, the oriented points are aligned iff they are simultaneously parallel and symmetric.

Parameters:

other another oriented point.

Returns:

whether the current oriented point and the given one are aligned.

See also:

isParallelTo, isSymmetricTo.

► **bool iSeeML::rob::OrPtConfig::isParallelTo (const OrPtConfig & *other*) const** `[inline]` Various method, checking whether two oriented points are parallel, ie whether their orientations are equal (modulo 2 pi).

Parameters:

other another oriented point.

Returns:

whether the current oriented point and the given one have same orientation.

See also:

orientation, iSeeML::Object::mod2pi, iSeeML::Object::isZero.

► **bool iSeeML::rob::OrPtConfig::isSymmetricTo (const OrPtConfig & *other*) const** `[inline]` Various method, checking whether two oriented points are symmetric, ie whether their orientations are symmetric wrt the line segment connecting their position. The vector connecting the positions and the vector of average orientation should be collinear: their vectorial product should be zero.

Parameters:

other another oriented point.

Returns:

whether the current oriented point and the given one are symmetric.

See also:

position, orientation, iSeeML::geom::Vector::operator^, iSeeML::Object::isZero.

► **bool iSeeML::rob::OrPtConfig::operator== (const OrPtConfig & *other*) const** `[inline]` Equality operator between oriented points.

Parameters:

other another oriented point.

Returns:

whether the two points are equal (same position and orientation).

See also:

position, iSeeML::geom::Point::operator==, isParallelTo.

► **OrPtConfig iSeeML::rob::OrPtConfig::opposite () const** `[inline]` Various method, giving the opposite oriented point of the current one: the opposite oriented point has the same position but opposite orientation (current oriented point's orientation $\pm \pi$).

Returns:

the opposite oriented point of the current one.

See also:

uTurn.

► **const double& iSeeML::rob::OrPtConfig::orientation () const** `[inline]` Description method, giving the orientation of the robot's main axis in this configuration.

Returns:

the orientation of the main axis.

Examples:

wxGuiFwd.cpp.

► **const iSeeML::geom::Point& iSeeML::rob::OrPtConfig::position () const** `[inline]` Description method, giving the position of the robot's reference point in this configuration.

Returns:

the position of the reference point.

Examples:

wxGuiFwd.cpp.

► **OrPtConfig& iSeeML::rob::OrPtConfig::uTurn ()** `[inline]` Modification method, turning the current oriented point to its opposite: the position does not change but the orientation is replaced by its opposite ($\pm \pi$ is added to it).

Returns:

the new oriented point, one transformed to its opposite.

► **void iSeeML::rob::OrPtConfig::writeTo (ostream & *O*) const** `[inline, virtual]` Description method, writing the configuration in a given output stream: coordinate in each dimension is written, between paranthesis and separated by commas.

Parameters:

O the output stream in which description is written.

See also:

algWriteTo.

Implements iSeeML::Object.

Reimplemented in iSeeML::rob::CurvConfig.

Member Data Documentation

► **const string iSeeML::rob::OrPtConfig::ClassName** `[static]` The class name is public, as this class can be instanced.

See also:

className.

Reimplemented from iSeeML::rob::Object.

Reimplemented in iSeeML::rob::CurvConfig.

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

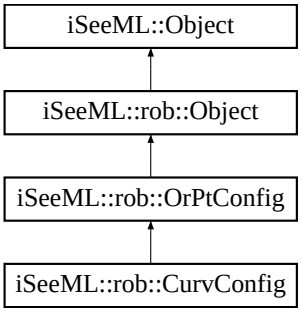
- include/iSeeML/rob/OrPtConfig.hpp

2.4.3.3 iSeeML::rob::CurvConfig Class Reference

This class defines a more precise configuration than classical ones for mobile robots.

`#include <CurvConfig.hpp>`

Inheritance diagram for iSeeML::rob::CurvConfig:



Public Member Functions

- CurvConfig ()

  The default constructor should only be used for array initializations: it correspond to a default oriented point with zero curvature.

- CurvConfig (const iSeeML::rob::OrPtConfig &OrPt, const double &kappa)

  The main constructor.

- CurvConfig (const iSeeML::geom::Point &P, const double &theta, const double &kappa)

  A first detailled constructor allows to give the classical configuration as a point and an orientation.

- CurvConfig (const double &x, const double &y, const double &theta, const double &kappa)

  A second detailled constructor allows to give the classical configuration as two Cartesian coordinates and an orientation.

- iSeeML::Object & clone () const

  Description method, giving a copy of the current configuration.

- void writeTo (ostream &O) const

  Description method, writing the configuration in a given output stream: coordinate in each dimension is written, between paranthesis and separated by commas.

- const double & curvature () const

  Description method, giving the intantaneous curvature, in this configuration, of the curve followed by the robot's reference point.

Static Public Attributes

- static const string ClassName

  The class name is public, as this class can be instanced.

Protected Member Functions

- int algDimension () const

  Description method, giving the dimension of the containing space (4) when this configuration is considered as an algebraic vector.

- double algCoord (const int i) const

  Description method, giving the coordinate of given index of the configuration (both coordinates of the position, then orientation, then curvature).

Detailed Description

This class defines a more precise configuration than classical ones for mobile robots. This configuration contains, added to the classical position of a reference point and orientation of a main axis, the curvature of the curve followed by the reference point (representing the directing wheels' orientation).

This class adds to those of iSeeML::rob::OrPtConfig a description method giving the curvature, and redefines description method writeTo.

Author:

Alexis Scheuer.

Version:

1.0

Constructor & Destructor Documentation

► **iSeeML::rob::CurvConfig::CurvConfig** () `[inline]` The default constructor should only be used for array initializations: it correspond to a default oriented point with zero curvature.

See also:

default constructor of iSeeML::rob::OrPtConfig.

► **iSeeML::rob::CurvConfig::CurvConfig** (**const iSeeML::rob::OrPtConfig &** ***OrPt*****, const double &** ***kappa***) `[inline]` The main constructor.

Parameters:

OrPt a classical configuration as an oriented point,
kappa the curvature of the reference point's curve.

► **iSeeML::rob::CurvConfig::CurvConfig (const iSeeML::geom::Point &** ***P*****, const double &** ***theta*****, const double &** ***kappa***) `[inline]` A first detailled constructor allows to give the classical configuration as a point and an orientation.

Parameters:

P the position of the reference point,
theta the orientation of the main axis,
kappa the curvature of the reference point's curve.

See also:

main constructor of iSeeML::rob::OrPtConfig.

► **iSeeML::rob::CurvConfig::CurvConfig (const double & *x*, const double & *y*, const double & *theta*, const double & *kappa*)** **`[inline]`** A second detailled constructor allows to give the classical configuration as two Cartesian coordinates and an orientation.

Parameters:

x the X coordinate of the reference point,
y the Y coordinate of the reference point,
theta the orientation of the main axis,
kappa the curvature of the reference point's curve.

See also:

usefull constructor of iSeeML::rob::OrPtConfig.

Member Function Documentation

► **double iSeeML::rob::CurvConfig::algCoord (const int *i*) const** **`[inline, protected, virtual]`** Description method, giving the coordinate of given index of the configuration (both coordinates of the position, then orientation, then curvature).

Parameters:

i the index of the searched coordinate.

Precondition:

the parameter is an index, and should be between one and four. If not, and `ISEEML_CHECK_CURV_CONFIG_PRECOND` is defined (see CompilerFlags.h), an error is generated.

Returns:

the double coordinate associated to the index if this one is between 1 and dimension, or 0.

See also:

curvature, iSeeML::rob::OrPtConfig::algCoord.

Reimplemented from iSeeML::rob::OrPtConfig.

► **int iSeeML::rob::CurvConfig::algDimension** () **const** `[inline, protected, virtual]` Description method, giving the dimension of the containing space (4) when this configuration is considered as an algebraic vector. This representation is mainly used to define a unified writeTo. Dimension cannot be static, as it is virtual.

Reimplemented from iSeeML::rob::OrPtConfig.

► **iSeeML::Object& iSeeML::rob::CurvConfig::clone** () **const** `[inline, virtual]` Description method, giving a copy of the current configuration. This clone is dynamically allocated (using default copy constructor), it has to be deleted later.

Returns:

a copy/clone of the current configuration.

Reimplemented from iSeeML::rob::OrPtConfig.

► **const double& iSeeML::rob::CurvConfig::curvature** () **const** `[inline]` Description method, giving the intantaneous curvature, in this configuration, of the curve followed by the robot's reference point.

Returns:

the intantaneous curvature of the robot's reference point's curve.

Examples:

wxGuiFwd.cpp.

► **void iSeeML::rob::CurvConfig::writeTo** (**ostream & *O***) **const** `[inline, virtual]` Description method, writing the configuration in a given output stream: coordinate in each dimension is written, between paranthesis and separated by commas.

Parameters:

O the output stream in which description is written.

See also:

algWriteTo.

Reimplemented from iSeeML::rob::OrPtConfig.

Member Data Documentation

► **const string iSeeML::rob::CurvConfig::ClassName** `[static]` The class name is public, as this class can be instanced.

See also:

className.

Reimplemented from iSeeML::rob::OrPtConfig.

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

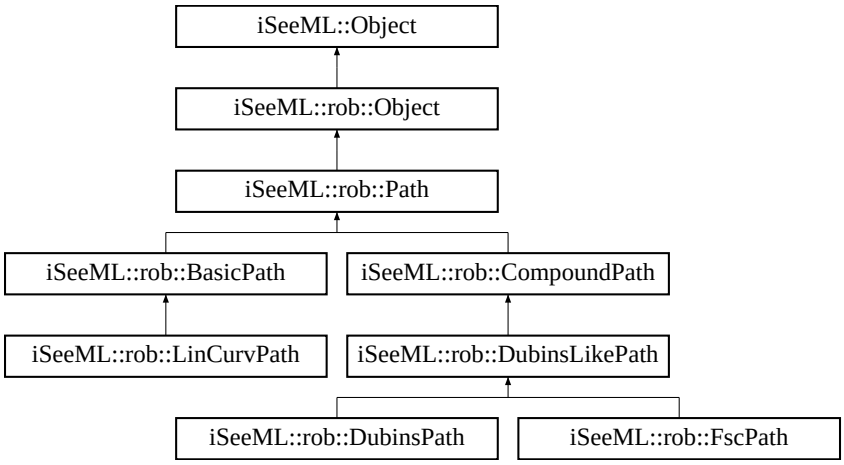
- include/iSeeML/rob/CurvConfig.hpp

2.4.3.4 iSeeML::rob::Path Class Reference

This class is the base class of all robotic paths in ISeeML.

`#include <Path.hpp>`

Inheritance diagram for iSeeML::rob::Path:



Public Member Functions

- virtual const iSeeML::rob::OrPtConfig & start () const =0

  Description method, giving the path's starting configuration.

- virtual const iSeeML::rob::OrPtConfig & end () const =0

Description method, giving the path's final configuration.

- virtual double length () const =0

  Description method, giving the path's length.

- virtual double deflection () const =0

  Description method, giving the path's deflection (change of orientation).

- bool operator< (const Path &other) const

  Order operator, defining an order relation on the paths' set, based on the path's length.

- virtual iSeeML::rob::CurvConfig operator[ ] (const double &s) const =0

  Description method, giving a configuration at a given arc length along the path.

Detailed Description

This class is the base class of all robotic paths in ISeeML. A path is the geometric aspect of a robot's motion. It is therefore a continuous set of configurations, and transitions between these configurations respect the kinematic constraints of this robot as, for example, continuity of the robot's orientation, bounds on the curvature (the inverse of the turning radius), continuity of the curvature, bounds on its derivative, etc.

This class contains the virtual methods common to all robotic paths' classes, which are only description methods, an order operator<(const Path&), which uses the virtual description method length, and a description operator, operator[ ](const double&), returning the configuration at a given arc length.

Note that starting and final configurations are referenced as classical configurations (to which a curvature configuration's reference is automatically transformed), while configurations at a given arc length are computed when needed, and are given as curvature configuration (to which a classical configuration is automatically transformed).

Author:

Alexis Scheuer.

Version:

1.0

Member Function Documentation

► **virtual double iSeeML::rob::Path::deflection () const** `[pure virtual]`
Description method, giving the path's deflection (change of orientation).

Returns:

the path's deflection.

Implemented in iSeeML::rob::CompoundPath, and iSeeML::rob::LinCurvPath.

► **virtual const iSeeML::rob::OrPtConfig& iSeeML::rob::Path::end** () **const** `[pure virtual]` Description method, giving the path's final configuration.

Returns:

the path's final configuration.

Implemented in iSeeML::rob::CompoundPath, and iSeeML::rob::LinCurvPath.

► **virtual double iSeeML::rob::Path::length** () **const** `[pure virtual]` Description method, giving the path's length.

Returns:

the path's length.

Implemented in iSeeML::rob::CompoundPath, and iSeeML::rob::LinCurvPath.

► **bool iSeeML::rob::Path::operator<** (**const Path &** ***other***) **const** `[inline]` Order operator, defining an order relation on the paths' set, based on the path's length. In other words, a path is smaller than a second one iff it is shorter (its length is smaller than the second one's). However, zero length paths are generally not correct: such a path is consider as shorter if and only if the second one is also a zero length path.

Parameters:

other another path.

Returns:

whether the current path is smaller (shorter) than the given one.

See also:

length, iSeeML::Object::isPositive.

► **virtual iSeeML::rob::CurvConfig iSeeML::rob::Path::operator[ ]** (**const double &** ***s***) **const** `[pure virtual]` Description method, giving a configuration at a given arc length along the path.

Parameters:

s the arc length.

Precondition:

the given arc length should be positive and less than the path's length.

Returns:

the configuration at given arc length along the path.

Implemented in iSeeML::rob::CompoundPath, and iSeeML::rob::LinCurvPath.

► **virtual const iSeeML::rob::OrPtConfig& iSeeML::rob::Path::start () const [pure virtual]** Description method, giving the path's starting configuration.

Returns:

the path's starting configuration.

Implemented in iSeeML::rob::CompoundPath, and iSeeML::rob::LinCurvPath.

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

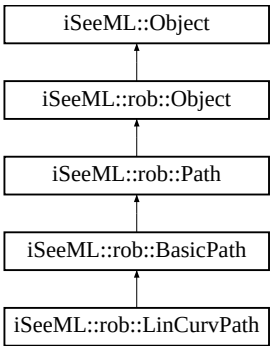
- include/iSeeML/rob/Path.hpp

2.4.3.5 iSeeML::rob::BasicPath Class Reference

This class defines superclass of basic paths, from which compound paths can be built.

`#include <BasicPath.hpp>`

Inheritance diagram for iSeeML::rob::BasicPath:



Detailed Description

This class defines superclass of basic paths, from which compound paths can be built.

Author:

Alexis Scheuer.

Version:

1.0

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

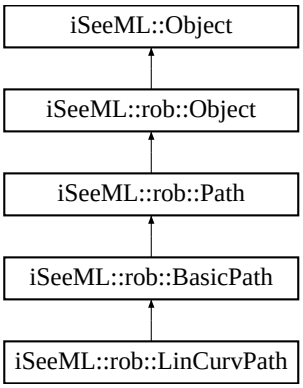
- include/iSeeML/rob/BasicPath.hpp

2.4.3.6 iSeeML::rob::LinCurvPath Class Reference

This class defines linear curvature elementary paths, for which the curvature derivative (with respect to the arc length) is constant.

`#include <LinCurvPath.hpp>`

Inheritance diagram for iSeeML::rob::LinCurvPath:



Public Member Functions

- LinCurvPath ()

  The default constructor should only be used for array initializations: it generates a path starting from default configuration, with constant curvature and zero length.

- LinCurvPath (const iSeeML::rob::CurvConfig &start, const double &sharpness=0, const double &length=0)

  The main constructor.

- const iSeeML::rob::CurvConfig & start () const

  Description method, giving the path's starting configuration.

- const iSeeML::rob::CurvConfig & end () const

  Description method, giving the path's final configuration (as a configuration for mobile robot with curvature).

- const double & sharpness () const

  Description method, giving the path's constant curvature derivative with respect to the arc length.

- double length () const

  Description method, giving the path's length.

- double deflection () const

  Description method, giving the path's deflection (change of orientation).

- iSeeML::Object & clone () const

  Description method, giving a copy of the current linear curvature path.

- void writeTo (ostream &O) const

  Description method, writing a description of the path in a given output stream.

- iSeeML::rob::CurvConfig operator[ ] (const double &s) const

  Description method, giving a configuration at a given arc length along the path.

Static Public Attributes

- static const string ClassName

  The class name is public, as this class can be instanced.

Detailed Description

This class defines linear curvature elementary paths, for which the curvature derivative (with respect to the arc length) is constant. These are the only elementary paths of ISeeML, the other paths being made of paths of this kind (they are compound paths).

Author:

Alexis Scheuer.

Version:

1.0

Examples:

wxGuiFwd.cpp.

Constructor & Destructor Documentation

► **iSeeML::rob::LinCurvPath::LinCurvPath** () `[inline]` The default constructor should only be used for array initializations: it generates a path starting from default configuration, with constant curvature and zero length.

See also:

default constructor of iSeeML::rob::CurvConfig.

► **iSeeML::rob::LinCurvPath::LinCurvPath (const iSeeML::rob::CurvConfig & *start*, const double & *sharpness* = 0, const double & *length* = 0)** `[inline]` The main constructor.

Parameters:

start the starting configuration,
sharpness the constant curvature derivative with respect to the arc length (default value is zero),
length the length of the path (default value is zero).

See also:

operator[].

Member Function Documentation

► **iSeeML::Object& iSeeML::rob::LinCurvPath::clone** () **const** `[inline, virtual]` Description method, giving a copy of the current linear curvature path. This clone is dynamically allocated (using default copy constructor), it has to be deleted later.

Returns:

a copy/clone of the current linear curvature path.

Implements iSeeML::Object.

► **double iSeeML::rob::LinCurvPath::deflection** () **const** `[inline, virtual]` Description method, giving the path's deflection (change of orientation).

Returns:

the path's deflection.

Implements iSeeML::rob::Path.

► **const iSeeML::rob::CurvConfig& iSeeML::rob::LinCurvPath::end** () **const** `[inline, virtual]` Description method, giving the path's final configuration (as a configuration for mobile robot with curvature).

Returns:

the path's final configuration.

Implements iSeeML::rob::Path.

Examples:

wxGuiFwd.cpp.

► **double iSeeML::rob::LinCurvPath::length** () **const** `[inline, virtual]` Description method, giving the path's length.

Returns:

the path's length.

Implements iSeeML::rob::Path.

Examples:

wxGuiFwd.cpp.

► **iSeeML::rob::CurvConfig iSeeML::rob::LinCurvPath::operator[ ] (const double &** ***s*****) const** `[virtual]` Description method, giving a configuration at a given arc length along the path.

Parameters:

s the arc length.

Precondition:

the given arc length should be positive and less than the path's length. If this is not true, an error message is generated if `ISEEML_CHECK_LIN_PATH_PRECOND` is defined, and the arc length is considered as zero if smaller and as path's length if bigger, if `ISEEML_CHECK_ARRAY_ELEMT` is defined (see CompilerFlags.h).

Returns:

the configuration at given arc length along the path.

Implements iSeeML::rob::Path.

► **const double& iSeeML::rob::LinCurvPath::sharpness () const `[inline]`** Description method, giving the path's constant curvature derivative with respect to the arc length.

Returns:

the path's curvature derivative.

Examples:

wxGuiFwd.cpp.

► **const iSeeML::rob::CurvConfig& iSeeML::rob::LinCurvPath::start () const `[inline, virtual]`** Description method, giving the path's starting configuration. (as a configuration for mobile robot with curvature).

Returns:

the path's starting configuration.

Implements iSeeML::rob::Path.

Examples:

wxGuiFwd.cpp.

► **void iSeeML::rob::LinCurvPath::writeTo (ostream & *O*) const `[inline, virtual]`** Description method, writing a description of the path in a given output stream. The starting configuration is given, then the path parameters (sharpness and length) between braces, then the final configuration, the whole between brackets.

Parameters:

O the output stream in which description is written.

Implements iSeeML::Object.

Member Data Documentation

► **const string iSeeML::rob::LinCurvPath::ClassName** `[static]` The class name is public, as this class can be instanced.

See also:

className.

Reimplemented from iSeeML::rob::BasicPath.

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

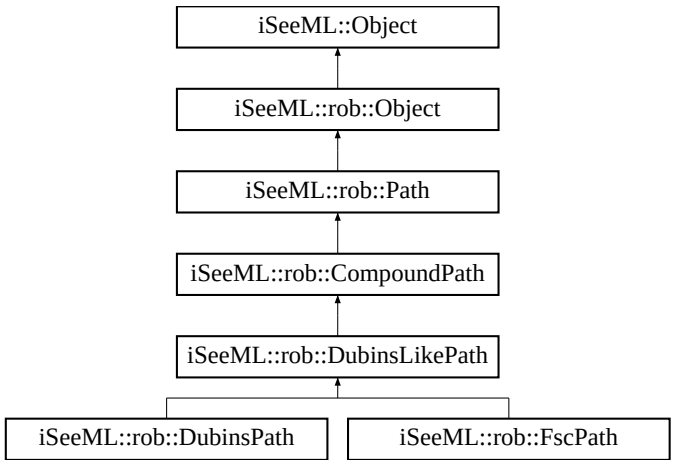
- include/iSeeML/rob/LinCurvPath.hpp

2.4.3.7 iSeeML::rob::CompoundPath Class Reference

This class defines compound (or complex, or composed) paths, which are made of a set of basic paths.

`#include <CompoundPath.hpp>`

Inheritance diagram for iSeeML::rob::CompoundPath:



Public Member Functions

- void writeTo (ostream &O) const

  Description method, writing a description of the path in a given output stream (the array of pieces is written).

- const iSeeML::rob::OrPtConfig & start () const

  Description method, giving the path's starting configuration (first piece's starting configuration).

- const iSeeML::rob::OrPtConfig & end () const

  Description method, giving the path's final configuration (last piece's starting configuration).

- virtual int nbPieces () const =0

  Description method, giving the number of pieces of the path.

- const iSeeML::rob::BasicPath & piece (const int index) const

  Description method, returning the (constant) basic path of given index in the list of which this compound path is made.

- double length () const

  Description method, giving the path's length (sum of the pieces' length).

- double deflection () const

  Description method, giving the path's deflection (sum of the pieces' deflection).

- iSeeML::rob::BasicPath & piece (const int index)

  Modification method, returning (for modification) the basic path of given index in the list of which this compound path is made.

- iSeeML::rob::CurvConfig operator[ ] (const double &s) const

  Description method, giving a configuration at a given arc length along the path.

Protected Member Functions

- iSeeML::rob::BasicPath & getPiece (const int index) const

  Method returning the basic path of given index in the list of which this compound path is made.

- virtual iSeeML::rob::BasicPath & _piece (const int index) const =0

  Virtual method returning the basic path of given index in the list of which this compound path is made (no verification).

Detailed Description

This class defines compound (or complex, or composed) paths, which are made of a set of basic paths. Nearly all virtual methods of iSeeML::rob::Path are defined in this class (except className and clone), sub-classes "just" have to define this one and constructors (this is not so easy, as it generally requires a planning method).

Author:

Alexis Scheuer.

Version:

1.0

Member Function Documentation

► **virtual iSeeML::rob::BasicPath& iSeeML::rob::CompoundPath::_piece (const int *index*) const** `[protected, pure virtual]` Virtual method returning the basic path of given index in the list of which this compound path is made (no verification).

Parameters:

index the index of the desired piece (between 1 and the result of nbPieces).

Returns:

the basic path whose index has been given.

See also:

nbPieces, piece, getPiece.

Implemented in iSeeML::rob::DubinsLikePath.

► **double iSeeML::rob::CompoundPath::deflection () const** `[inline, virtual]` Description method, giving the path's deflection (sum of the pieces' deflection).

Returns:

the path's deflection.

See also:

piece, BasicPath::deflection.

Implements iSeeML::rob::Path.

► **const iSeeML::rob::OrPtConfig& iSeeML::rob::CompoundPath::end () const** **`[inline, virtual]`** Description method, giving the path's final configuration (last piece's starting configuration).

Returns:

the path's final configuration.

See also:

piece, BasicPath::end.

Implements iSeeML::rob::Path.

Examples:

wxGuiFwd.cpp.

► **iSeeML::rob::BasicPath& iSeeML::rob::CompoundPath::getPiece (const int *index*) const** **`[inline, protected]`** Method returning the basic path of given index in the list of which this compound path is made.

Parameters:

index the index of the desired piece (between 1 and the result of nbPieces).

Precondition:

parameter is an index, and should be in the correct interval. An error message is generated if this index is not correct and `ISEEML_CHECK_COMP_PATH_PRECOND` is defined, and index is corrected (to one if smaller, to nbPieces if bigger) if `ISEEML_CHECK_ARRAY_ELEMT` is defined (see CompilerFlags.h).

Returns:

the basic path whose index has been given.

See also:

nbPieces, piece, _piece.

► **double iSeeML::rob::CompoundPath::length () const** **`[inline, virtual]`** Description method, giving the path's length (sum of the pieces' length).

Returns:

the path's length.

See also:

piece, BasicPath::length.

Implements iSeeML::rob::Path.

Examples:

LengthFwd.cpp.

► **virtual int iSeeML::rob::CompoundPath::nbPieces () const [pure virtual]** Description method, giving the number of pieces of the path.

Returns:

the number of pieces of the path.

Implemented in iSeeML::rob::DubinsLikePath.

► **iSeeML::rob::CurvConfig iSeeML::rob::CompoundPath::operator[ ] (const double & *s*) const [inline, virtual]** Description method, giving a configuration at a given arc length along the path.

Parameters:

s the arc length.

Precondition:

the given arc length should be positive and less than the path's length. If this is not true, an error message is generated if `ISEEML_CHECK_COMP_PATH_PRECOND` is defined, and the arc length is considered as zero if smaller and as path's length if bigger, if `ISEEML_CHECK_ARRAY_ELEMT` is defined (see CompilerFlags.h).

Returns:

the configuration at given arc length along the path.

See also:

nbPieces, length, piece, BasicPath::operator[ ].

Implements iSeeML::rob::Path.

► **iSeeML::rob::BasicPath& iSeeML::rob::CompoundPath::piece (const int *index*) `[inline]`** Modification method, returning (for modification) the basic path of given index in the list of which this compound path is made.

Parameters:

index the index of the basic path to change (between 1 and the result of nbPieces).

Precondition:

parameter is an index, and should be in the correct interval. An error message is generated if this index is not correct and `ISEEML_CHECK_COMP_PATH_PRECOND` is defined, and index is corrected (to one if smaller, to nbPieces if bigger) if `ISEEML_CHECK_ARRAY_ELEMT` is defined (see CompilerFlags.h).

See also:

nbPieces.

► **const iSeeML::rob::BasicPath& iSeeML::rob::CompoundPath::piece (const int *index*) const `[inline]`** Description method, returning the (constant) basic path of given index in the list of which this compound path is made. This method, as well as the other `piece` method, calls a protected one (`getPiece`).

Parameters:

index the index of the desired piece (between 1 and the result of nbPieces).

Precondition:

parameter is an index, and should be in the correct interval. An error message is generated if this index is not correct and `ISEEML_CHECK_COMP_PATH_PRECOND` is defined, and index is corrected (to one if smaller, to nbPieces if bigger) if `ISEEML_CHECK_ARRAY_ELEMT` is defined (see CompilerFlags.h).

Returns:

the piece of the path whose index has been given.

See also:

nbPieces.

► **const iSeeML::rob::OrPtConfig& iSeeML::rob::CompoundPath::start () const `[inline, virtual]`** Description method, giving the path's starting configuration (first piece's starting configuration).

Returns:

the path's starting configuration.

See also:

piece, BasicPath::start.

Implements iSeeML::rob::Path.

Examples:

wxGuiFwd.cpp.

► **void iSeeML::rob::CompoundPath::writeTo (ostream & *O*) const** **[inline, virtual]** Description method, writing a description of the path in a given output stream (the array of pieces is written). The array of basic paths is written between accolades ('{' and '}'), paths being separated by commas.

Parameters:

O the output stream in which description is written.

See also:

nbPieces, piece.

Implements iSeeML::Object.

Reimplemented in iSeeML::rob::DubinsPath, and iSeeML::rob::FscPath.

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

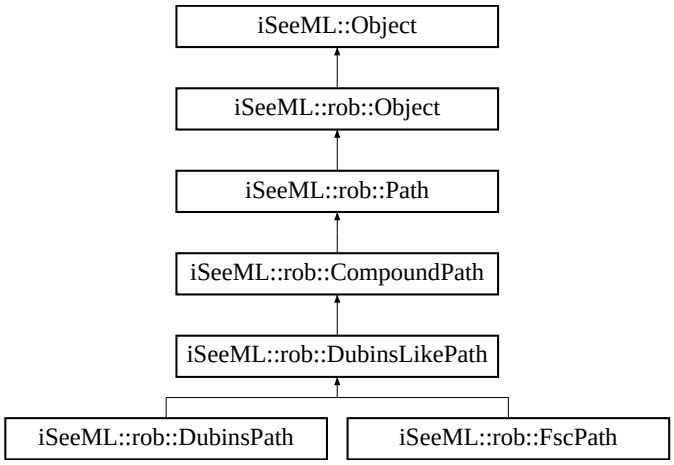
- include/iSeeML/rob/CompoundPath.hpp

2.4.3.8 iSeeML::rob::DubinsLikePath Class Reference

This class defines Dubins like paths, which are similar to Dubins paths.

`#include <DubinsLikePath.hpp>`

Inheritance diagram for iSeeML::rob::DubinsLikePath:



Public Types

- enum Type {

  lsl, lsr, rsl, rsr,

  lrl, rlr }

  Dubins-like paths can be of six types: their first and last parts are turns, their middle part is tangent to the first and last parts but can be a line segment or a turn.

- enum { nbPossiblePaths = 6, nbTurningCircles = 4 }

  Some methods will need these integer values.

Public Member Functions

- int nbPieces () const

  Description method, giving the number of pieces of the path.

- const Type & type () const

  Description method, giving the path's type (from lsl to rlr).

- const double & maxCurv () const

  Description method, giving the path's maximum curvature.

- const iSeeML::rob::LinCurvPath & lcPiece (const int index) const

Description method, returning the (linear curvature) piece of the path with a given index.

- virtual double turnRadius () const =0

  Description method, giving the radius of the turning circle (the circle of all the configurations which can be reached from a fixed one).

- void connect (const iSeeML::rob::OrPtConfig &start, const iSeeML::rob::OrPtConfig &goal)

  Modification method, defining type and pieces of a Dubins' like path in order to connect two given configurations with the shortest length (between nbPossiblePaths possibilities).

- DubinsLikePath & operator= (const DubinsLikePath &other)

  Copy operator (default one is not correct).

- virtual void computeCenters (const iSeeML::rob::OrPtConfig &start, const iSeeML::rob::OrPtConfig &goal, iSeeML::geom::Point centers[nbTurningCircles]) const =0

  Various method, computing the set of circle centers used to turn from starting configuration of the current path, or to turn to reach the given goal configuration.

Protected Member Functions

- DubinsLikePath (const Type &type, const double &maxCurv)

  The structure can be filled using this constructor, which is protected (only defined for sub-classes).

- DubinsLikePath (const DubinsLikePath &other)

  The copy constructor (the default one is not correct).

- ~DubinsLikePath ()

  The destructor (the default one is not correct).

- virtual int turnNbPieces (const double &length) const =0

  Description method, computing the number of pieces needed for a turn of given length.

- virtual bool getConnection (const double &dist, double &length, double &angle) const =0

  Description method, trying to compute the length of the line segment connecting the turning circles and the angle between the segment connecting the turning centers and the previous one.

- iSeeML::rob::LinCurvPath & lcPiece (const int index)

  Modification method, returning for modification the (linear curvature) piece of the path with a given index.

- void setNoPiece (const iSeeML::rob::OrPtConfig &start)

  Modification method, setting pieces of a non-valid Dubins' like path from a starting configuration (pieces array will only contain a zero length line segment).

- void definePieces (const iSeeML::rob::OrPtConfig &start, const double &defl1, const double &lenDefl, const double &defl3)
- virtual void addTurn (int &index, const iSeeML::rob::OrPtConfig $**$start, const double &defl)=0

  Modification method, adding to the pieces of a path a turn starting at a given configuration, with a given curvature sign and length.

- void addPiece (int &index, const iSeeML::rob::CurvConfig &start, const double &curvDeriv, const double &length)

  Modification method, adding to the pieces of a path a piece starting at a given configuration (with curvature), with a given curvature's derivative and length.

- iSeeML::rob::BasicPath & _piece (const int index) const

  Virtual method returning the basic path of given index in the list of which this compound path is made (no verification).

- virtual DubinsLikePath $*$ connectArray () const =0

  Various method, returning an array of nbPossiblePaths Dubins-like paths, initialised as clones of the current path.

- virtual DubinsLikePath & getSolution (DubinsLikePath $*$paths, const int index) const =0

  Various method, getting in an array of Dubins-like paths the element of given index.

Static Protected Member Functions

- static int turnSign (const int number, const Type &type)

  Description method, giving the sign (1 for left, -1 for right, 0 for a segment) of the turn or segment whose number is given for a path of given type.

Detailed Description

This class defines Dubins like paths, which are similar to Dubins paths. A Dubins like path is made of at most three parts, which can be a turn or a straight line. Dubins paths' parts are circular arcs or line segments, with a ponctual change of curvature at each part's end, while FSC paths' turns start and finish with zero curvature. Each part is represented by one to three pieces, which are linear curvature paths.

Author:

Alexis Scheuer.

Version:

1.0

Examples:

wxGuiFwd.cpp.

Member Enumeration Documentation

► **anonymous enum** Some methods will need these integer values.

Enumerator:

nbPossiblePaths the number of possible paths (6).

See also:

Type.

nbTurningCircles the number of usefull turning circles (4).

► **enum iSeeML::rob::DubinsLikePath::Type** Dubins-like paths can be of six types: their first and last parts are turns, their middle part is tangent to the first and last parts but can be a line segment or a turn. Thus, Dubins-like paths' types can be named by three letters: the first and third are *l* or *r* (Left or Right turn) and the second is *l*, *r* or *s* (Left or Right turn, or Segment).

Enumerator:

lsl Path made of a left turn, a straight line and a left turn.
lsr Path made of a left turn, a straight line and a right turn.
rsl Path made of a right turn, a straight line and a left turn.
rsr Path made of a right turn, a straight line and a right turn.
lrl Path made of a left turn, a right turn and a left turn.
rlr Path made of a right turn, a left turn and a right turn.

Constructor & Destructor Documentation

► **iSeeML::rob::DubinsLikePath::DubinsLikePath (const Type & *type*, const double & *maxCurv*) `[inline, protected]`** The structure can be filled using this constructor, which is protected (only defined for sub-classes). No other constructor is defined, as this class is virtual.

Parameters:

type the type of Dubins' path,
maxCurv the maximum curvature along the path (taken in absolute value).

► **iSeeML::rob::DubinsLikePath::DubinsLikePath (const DubinsLikePath & *other*) `[inline, protected]`** The copy constructor (the default one is not correct). The array of pieces of the other path is copied.

Parameters:

other the copied path.

► **iSeeML::rob::DubinsLikePath::~DubinsLikePath () `[inline, protected]`** The destructor (the default one is not correct). The array of pieces of the other path is freed.

Member Function Documentation

► **iSeeML::rob::BasicPath& iSeeML::rob::DubinsLikePath::_piece (const int *index*) const `[inline, protected, virtual]`** Virtual method returning the basic path of given index in the list of which this compound path is made (no verification).

Parameters:

index the index of the desired piece (between 1 and the result of nbPieces).

Returns:

the basic path whose index has been given.

See also:

nbPieces, piece, getPiece.

Implements iSeeML::rob::CompoundPath.

► **void iSeeML::rob::DubinsLikePath::addPiece (int & *index*, const iSeeML::rob::CurvConfig & *start*, const double & *curvDeriv*, const double & *length*) [inline, protected]** Modification method, adding to the pieces of a path a piece starting at a given configuration (with curvature), with a given curvature's derivative and length. Index `index` of the first undefined piece in the pieces array is incremented by one in this method, to remain the index of the first undefined piece in the array. Configuration `start` remains constant.

Parameters:

index the first free index in the pieces array (`index` will be added at this index, which will be incremented),
start the starting configuration,
curvDeriv the turn's curvature's derivative,
length the turn's length.

► **virtual void iSeeML::rob::DubinsLikePath::addTurn (int & *index*, const iSeeML::rob::OrPtConfig ** *start*, const double & *defl*) [protected, pure virtual]** Modification method, adding to the pieces of a path a turn starting at a given configuration, with a given curvature sign and length. Index `index` of the first undefined piece in the pieces array is modified by this method, to remain the index of the first undefined piece in the array. Adress `start` of the next starting configuration is also modified along this method. Maximum curvature of the current Dubins-like path is used in this method.

Parameters:

index the first free index in the pieces array (pieces will be added at this index and after, and the index will be incremented),
start a reference to the reference of the starting configuration (it will be changed to a ref. to the ref. of next starting config.),
defl the turn's deflection (change of orientation), giving the curvature sign.

Precondition:

This method only works if the maximum curvature is not zero (otherwise, it writes an error on `stderr`).

Implemented in iSeeML::rob::DubinsPath, and iSeeML::rob::FscPath.

► **virtual void iSeeML::rob::DubinsLikePath::computeCenters (const iSeeML::rob::OrPtConfig & *start*, const iSeeML::rob::OrPtConfig & *goal*, iSeeML::geom::Point *centers*[nbTurningCircles]) const [pure virtual]** Various method, computing the set of circle centers used to turn from starting configuration of the current path, or to turn to reach the given goal configuration.

Parameters:

start the starting configuration,
goal the configuration to reach,
centers the set of nbTurningCircles circle centers computed.

Precondition:

This method only works if the maximum curvature is not zero (zero curvature case is to be treated separately in constructors).

Implemented in iSeeML::rob::DubinsPath, and iSeeML::rob::FscPath.

Examples:

wxGuiFwd.cpp.

► **void iSeeML::rob::DubinsLikePath::connect (const iSeeML::rob::OrPtConfig & *start*, const iSeeML::rob::OrPtConfig & *goal*)** Modification method, defining type and pieces of a Dubins' like path in order to connect two given configurations with the shortest length (between nbPossiblePaths possibilities). Maximum curvature of the current Dubins-like path is used.

Parameters:

start the starting configuration,
goal the goal configuration.

► **virtual DubinsLikePath∗ iSeeML::rob::DubinsLikePath::connectArray () const `[protected, pure virtual]`** Various method, returning an array of nbPossiblePaths Dubins-like paths, initialised as clones of the current path.

Returns:

the array of Dubins-like paths.

Implemented in iSeeML::rob::DubinsPath, and iSeeML::rob::FscPath.

► **void iSeeML::rob::DubinsLikePath::definePieces (const iSeeML::rob::OrPtConfig & *start*, const double & *defl1*, const double & *lenDefl*, const double & *defl3*) `[protected]`** Modification method, defining pieces of a Dubins-like path from a starting configuration and deflections (change of orientation, for turns) or length (for straight lines) of each part.

Warning:

type and maximum curvature (at least) of the current path are used.

Parameters:

start the starting configuration,
defl1 the deflection of the first part of the path,
lenDefl the length or deflection of the second part,
defl3 the deflection of the third part of the path.

Precondition:

memory should not be full. If memory is full and `ISEEML_CHECK_NIL_POINTER` is defined (see CompilerFlags.h), an error message is generated (in this method or in setNoPiece) and the program exits.

Postcondition:

Number of pieces needed is estimated for memory allocation. Pieces array is then filled using setNoPiece, addTurn and addPiece. Coherence between estimated and used number of pieces is verified if `ISEEML_CHECK_DLIKE_PATH_POSTCOND` is defined (see CompilerFlags.h).

See also:

setNoPiece, addTurn, addPiece.

► **virtual bool iSeeML::rob::DubinsLikePath::getConnection (const double & *dist*, double & *length*, double & *angle*) const** **`[protected, pure virtual]`**
Description method, trying to compute the length of the line segment connecting the turning circles and the angle between the segment connecting the turning centers and the previous one. The angle can only be computed if the square distance between the turning centers is greater than a value depending of the path's type. Length is not computed for lrl or rlr path's type, and is negative when previous angle cannot be computed.

Parameters:

dist the distance between the turning centers,
length the length to compute (modified by the method),
angle the angle to compute (modified by the method).

Returns:

whether the computation was succesful or not (whether the distance `dist2` is greater than a value depending of the path's type).

Implemented in iSeeML::rob::DubinsPath, and iSeeML::rob::FscPath.

► **virtual DubinsLikePath& iSeeML::rob::DubinsLikePath::getSolution (DubinsLikePath * *paths*, const int *index*) const** **`[protected, pure virtual]`**
Various method, getting in an array of Dubins-like paths the element of given index.

This method is virtual as the **real** type of the paths in the first parameter array is the same as the real type of the current Dubins-like path (either Dubins' or FSC path). Size of these paths' representation being different, access in an array needs different computations.

Parameters:

paths the array of Dubins-like paths,
index the index of the searched path.

Precondition:

first parameter should be an array of paths obtained using connectArray.
second parameter should be a correct index for first parameter array (between 0 and nbPossiblePaths - 1).

Returns:

the array of Dubins-like paths.

See also:

connectArray.

Implemented in iSeeML::rob::DubinsPath, and iSeeML::rob::FscPath.

► **iSeeML::rob::LinCurvPath& iSeeML::rob::DubinsLikePath::lcPiece (const int *index*) `[inline, protected]`** Modification method, returning for modification the (linear curvature) piece of the path with a given index.

Parameters:

index the index of the desired piece (between 1 and the result of nbPiece()).

Precondition:

the given index should be between one and the result of nbPieces.

Returns:

the (modifiable) piece of the path whose index has been given.

See also:

piece.

► **const iSeeML::rob::LinCurvPath& iSeeML::rob::DubinsLikePath::lcPiece (const int *index*) const** `[inline]` Description method, returning the (linear curvature) piece of the path with a given index.

Parameters:

index the index of the desired piece (between 1 and the result of nbPiece()).

Precondition:

the given index should be between one and the result of nbPieces.

Returns:

the piece of the path whose index has been given.

See also:

piece.

Examples:

wxGuiFwd.cpp.

► **const double& iSeeML::rob::DubinsLikePath::maxCurv () const** `[inline]` Description method, giving the path's maximum curvature.

Returns:

the path's maximum curvature.

► **int iSeeML::rob::DubinsLikePath::nbPieces () const** `[inline, virtual]` Description method, giving the number of pieces of the path.

Returns:

the number of pieces of the path.

Implements iSeeML::rob::CompoundPath.

Examples:

wxGuiFwd.cpp.

► **DubinsLikePath& iSeeML::rob::DubinsLikePath::operator= (const DubinsLikePath & *other*) `[inline]`** Copy operator (default one is not correct).

Parameters:

other the path to copy.

Precondition:

memory should not be full. If memory is full and `ISEEML_CHECK_NULL_POINTER` is defined (see CompilerFlags.h), an error message is generated and the program exits.

Returns:

the current path, after modification.

► **void iSeeML::rob::DubinsLikePath::setNoPiece (const iSeeML::rob::OrPtConfig & *start*) `[inline, protected]`** Modification method, setting pieces of a non-valid Dubins' like path from a starting configuration (pieces array will only contain a zero length line segment).

Parameters:

start the starting configuration.

Precondition:

memory should not be full. If memory is full and `ISEEML_CHECK_NIL_POINTER` is defined (see CompilerFlags.h), an error message is generated and the program exits.

► **virtual int iSeeML::rob::DubinsLikePath::turnNbPieces (const double & *length*) const `[protected, pure virtual]`** Description method, computing the number of pieces needed for a turn of given length.

Parameters:

length the turn's length.

Returns:

the number of needed pieces.

Implemented in iSeeML::rob::DubinsPath, and iSeeML::rob::FscPath.

► **virtual double iSeeML::rob::DubinsLikePath::turnRadius** () **const** `[pure virtual]` Description method, giving the radius of the turning circle (the circle of all the configurations which can be reached from a fixed one). Maximum curvature of the current path is used. It should not be zero, or this method will return infinity.

Returns:

the turning circle's radius.

Implemented in iSeeML::rob::DubinsPath, and iSeeML::rob::FscPath.

Examples:

wxGuiFwd.cpp.

► **static int iSeeML::rob::DubinsLikePath::turnSign** (**const int** ***number***, **const Type &** ***type***) `[inline, static, protected]` Description method, giving the sign (1 for left, -1 for right, 0 for a segment) of the turn or segment whose number is given for a path of given type. As an example, a **lsr** type path has respective parts signs 1, 0 and -1, when a **rlr** type path has respective parts signs -1, 1 and -1.

Parameters:

number the number of the part whose sign is wanted,
type the type of the considered path.

Precondition:

the given number of the part should be between 1 and 3. If not, an error message is generated if `ISEEML_CHECK_DLIKE_PATH_PRECOND` is defined, and the number is considered as 1 if smaller and 3 if bigger, if `ISEEML_CHECK_ARRAY_ELEMT` is defined (see CompilerFlags.h).

Returns:

the part's sign.

► **const Type& iSeeML::rob::DubinsLikePath::type** () **const** `[inline]` Description method, giving the path's type (from lsl to rlr).

Returns:

the path's type.

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

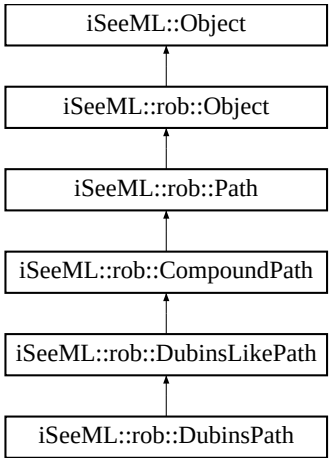
- include/iSeeML/rob/DubinsLikePath.hpp

2.4.3.9 iSeeML::rob::DubinsPath Class Reference

This class defines Dubins' paths, made of circular arcs tangentially connected by line segments, with no back-up maneuvres, corresponding to an optimal forward-only motion with a lower bounded turning radius.

`#include <DubinsPath.hpp>`

Inheritance diagram for iSeeML::rob::DubinsPath:



Public Member Functions

- DubinsPath ()

  The default constructor should only be used for array initializations: it generates a path starting from default oriented point, with type lsl, and zero maximum curvature and lengths.

- DubinsPath (const iSeeML::rob::OrPtConfig &start, const Type &type, const double &maxCurv, const double &length1, const double &length2, const double &length3)

  The 'forward' constructor: a Dubins' path is built from its starting oriented point, its type, its maximum curvature and its parts' lengths.

- DubinsPath (const iSeeML::rob::OrPtConfig &start, const iSeeML::rob::OrPtConfig &goal, const double &maxCurv)

  The 'goto' constructor: a Dubins' path is built, starting from a given iSeeML::rob::OrPtConfig and reaching an other one with a maximum curvature.

- DubinsPath (const DubinsPath &other)

The copy constructor.

- iSeeML::Object & clone () const

  Description method, giving a copy of the current Dubins' path.

- void writeTo (ostream &O) const

  Description method, writing a description of the path in a given output stream.

- double turnRadius () const

  Description method, giving the radius of the turning circle (the circle of all the configurations which can be reached from a fixed one).

- DubinsPath & operator= (const DubinsPath &other)

  Copy operator (default one is not correct).

- void computeCenters (const iSeeML::rob::OrPtConfig &start, const iSeeML::rob::OrPtConfig &goal, iSeeML::geom::Point centers[nbTurningCircles]) const

  Various method, computing the set of circle centers used to turn from starting configuration of the current path, or to turn to reach the given goal configuration.

Static Public Attributes

- static const string ClassName

  The class name is public, as this class can be instanced.

Protected Member Functions

- int turnNbPieces (const double &defl) const

  Description method, computing the number of pieces (0 or 1) needed for a turn of given deflection.

- virtual bool getConnection (const double &dist, double &length, double &angle) const

  Description method, trying to compute the length of the line segment connecting the turning circles and the angle between the segment connecting the turning centers and the previous one.

- void addTurn (int &index, const iSeeML::rob::OrPtConfig **start, const double &defl)

Modification method, adding to the pieces of a path a turn starting at a given configuration, with a given curvature sign and length.

- iSeeML::rob::DubinsLikePath ∗ connectArray () const

  Various method, returning an array of nbPossiblePaths Dubins-like paths, initialised as clones of the current path.

- iSeeML::rob::DubinsLikePath & getSolution (iSeeML::rob::DubinsLikePath ∗paths, const int index) const

  Various method, getting in an array of Dubins-like paths the element of given index.

Detailed Description

This class defines Dubins' paths, made of circular arcs tangentially connected by line segments, with no back-up maneuvres, corresponding to an optimal forward-only motion with a lower bounded turning radius.

See also:

iSeeML::rob::FscPath.

Author:

Alexis Scheuer.

Version:

1.0

Examples:

Fwd.cpp, LengthFwd.cpp, TimeFwd.cpp, and wxGuiFwd.cpp.

Constructor & Destructor Documentation

► **iSeeML::rob::DubinsPath::DubinsPath** () **`[inline]`** The default constructor should only be used for array initializations: it generates a path starting from default oriented point, with type lsl, and zero maximum curvature and lengths.

See also:

default constructor of iSeeML::rob::DubinsLikePath.

► **iSeeML::rob::DubinsPath::DubinsPath (const iSeeML::rob::OrPtConfig & *start*, const Type & *type*, const double & *maxCurv*, const double & *length1*, const double & *length2*, const double & *length3*)** `[inline]` The 'forward' constructor: a Dubins' path is built from its starting oriented point, its type, its maximum curvature and its parts' lengths.

Parameters:

start the starting configuration,
type the type of Dubins' path,
maxCurv the maximum curvature along the path (taken in absolute value),
length1 the length of the first part of the path,
length2 the length of the second part of the path,
length3 the length of the third part of the path.

► **iSeeML::rob::DubinsPath::DubinsPath (const iSeeML::rob::OrPtConfig & *start*, const iSeeML::rob::OrPtConfig & *goal*, const double & *maxCurv*)** `[inline]` The 'goto' constructor: a Dubins' path is built, starting from a given iSeeML::rob::OrPtConfig and reaching an other one with a maximum curvature.

Parameters:

start the starting configuration,
goal the goal configuration,
maxCurv the maximum curvature along the path.

► **iSeeML::rob::DubinsPath::DubinsPath (const DubinsPath & *other*)** `[inline]` The copy constructor.

Parameters:

other the copied path.

Member Function Documentation

► **void iSeeML::rob::DubinsPath::addTurn (int & *index*, const iSeeML::rob::OrPtConfig ** *start*, const double & *defl*)** `[protected, virtual]` Modification method, adding to the pieces of a path a turn starting at a given configuration, with a given curvature sign and length. Maximum curvature of the current Dubins' path is used.

Parameters:

index the first free index in the pieces array (pieces will be added at this index and after, and the index will be incremented),
start a reference to the reference of the starting configuration (it will be changed to a ref. to the ref. of next starting config.),

defl the turn's deflection (change of orientation), giving the curvature sign.

Precondition:

the maximum curvature should not be zero. If it is, an error message is generated if `ISEEML_CHECK_DUBINS_PATH_PRECOND` is defined (see CompilerFlags.h).

Implements iSeeML::rob::DubinsLikePath.

► **iSeeML::Object& iSeeML::rob::DubinsPath::clone () const `[inline, virtual]`** Description method, giving a copy of the current Dubins' path. This clone is dynamically allocated (using copy constructor), it has to be deleted later.

Returns:

a copy/clone of the current Dubins' path.

Implements iSeeML::Object.

► **void iSeeML::rob::DubinsPath::computeCenters (const iSeeML::rob::OrPtConfig & *start*, const iSeeML::rob::OrPtConfig & *goal*, iSeeML::geom::Point *centers*[nbTurningCircles]) const `[virtual]`** Various method, computing the set of circle centers used to turn from starting configuration of the current path, or to turn to reach the given goal configuration. Circles' radiuses are the opposite of the maximum curvature of the current path (maximum curvature should not be zero).

Parameters:

start the starting configuration,
goal the configuration to reach,
centers the set of circle centers computed.

Precondition:

the maximum curvature should not be zero. If it is, an error message is generated if `ISEEML_CHECK_DUBINS_PATH_PRECOND` is defined (see CompilerFlags.h).

Implements iSeeML::rob::DubinsLikePath.

► **iSeeML::rob::DubinsLikePath* iSeeML::rob::DubinsPath::connectArray () const `[inline, protected, virtual]`** Various method, returning an array of nbPossiblePaths Dubins-like paths, initialised as clones of the current path.

Returns:

the array of Dubins-like paths.

Implements iSeeML::rob::DubinsLikePath.

► **virtual bool iSeeML::rob::DubinsPath::getConnection (const double & *dist*, double & *length*, double & *angle*) const** `[protected, virtual]` Description method, trying to compute the length of the line segment connecting the turning circles and the angle between the segment connecting the turning centers and the previous one. The angle can only be computed if the square distance between the turning centers is greater than a value depending of the path's type. Length is not computed for **lrl** or **rlr** path's type, and is negative when previous angle cannot be computed.

Parameters:

dist the distance between the turning centers,
length the length to compute (modified by the method),
angle the angle to compute (modified by the method).

Returns:

whether the computed was succesful or not (whether the distance `dist2` is greater than a value depending of the path's type).

Implements iSeeML::rob::DubinsLikePath.

► **iSeeML::rob::DubinsLikePath& iSeeML::rob::DubinsPath::getSolution (iSeeML::rob::DubinsLikePath * *paths*, const int *index*) const** `[inline, protected, virtual]` Various method, getting in an array of Dubins-like paths the element of given index.

Parameters:

paths the array of Dubins-like paths,
index the index of the searched path.

Precondition:

first parameter should be an array of paths obtained using connectArray.
second parameter should be a correct index for first parameter array (between 0 and nbPossiblePaths - 1). If not, an error message is generated if `ISEEML_CHECK_DUBINS_PATH_PRECOND` is defined, and the number is considered as 1 if smaller and 3 if bigger, if `ISEEML_CHECK_ARRAY_ELEMT` is defined (see CompilerFlags.h).

Returns:

the array of Dubins-like paths.

Implements iSeeML::rob::DubinsLikePath.

► **DubinsPath& iSeeML::rob::DubinsPath::operator= (const DubinsPath & *other*) `[inline]`** Copy operator (default one is not correct).

Parameters:

other the path to copy.

Returns:

the current path, after modification.

► **int iSeeML::rob::DubinsPath::turnNbPieces (const double & *defl*) const `[inline, protected, virtual]`** Description method, computing the number of pieces (0 or 1) needed for a turn of given deflection.

Parameters:

defl the turn's deflection.

Returns:

the number of needed pieces, which is 0 or 1.

Implements iSeeML::rob::DubinsLikePath.

► **double iSeeML::rob::DubinsPath::turnRadius () const `[inline, virtual]`** Description method, giving the radius of the turning circle (the circle of all the configurations which can be reached from a fixed one).

Warning:

This method returns infinity if the maximum curvature is zero (zero curvature path should not use it).

Returns:

the turning circle's radius.

Implements iSeeML::rob::DubinsLikePath.

► **void iSeeML::rob::DubinsPath::writeTo (ostream & *O*) const `[virtual]`** Description method, writing a description of the path in a given output stream. Starting configuration (as an oriented point) is given, followed by a short description of each part (curvature and length), and by the final configuration.

Parameters:

O the output stream in which description is written.

Reimplemented from iSeeML::rob::CompoundPath.

Member Data Documentation

► **const string iSeeML::rob::DubinsPath::ClassName `[static]`** The class name is public, as this class can be instanced.

See also:

className.

Reimplemented from iSeeML::rob::DubinsLikePath.

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

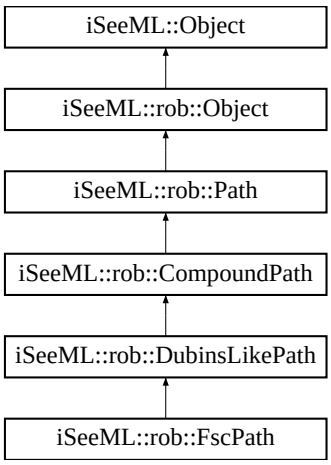
- include/iSeeML/rob/DubinsPath.hpp

2.4.3.10 iSeeML::rob::FscPath Class Reference

Objects of this class are Forward-only Sub-optimal Continuous-curvature (FSC) paths, made of circular arcs, pieces of clothoid and line segments.

`#include <FscPath.hpp>`

Inheritance diagram for iSeeML::rob::FscPath:



Public Member Functions

- FscPath ()

  The default constructor should only be used for array initializations: it generates a path starting from default oriented point, with type lsl, and zero maximum curvature, maximum curvature's derivative and lengths.

- FscPath (const iSeeML::rob::OrPtConfig &start, const Type &type, const double &maxCurv, const double &maxCDer, const double &length1, const double &length2, const double &length3)

  The 'forward' constructor: a FSC path is built from its starting oriented point, its type, its maximum curvature, maximum curvature's derivative and parts' lengths.

- FscPath (const iSeeML::rob::OrPtConfig &start, const Type &type, const double &maxCurv, const double &maxCDer, const double &limDefl, const double &turnRad, const double &turnAng, const double &length1, const double &length2, const double &length3)

  The complete 'forward' constructor: a FSC path is built from its starting oriented point, its type, its maximum curvature, maximum curvature's derivative, limit deflection, turning radius, turning angle, and parts' lengths.

- FscPath (const iSeeML::rob::OrPtConfig &start, const iSeeML::rob::OrPtConfig &goal, const double &maxCurv, const double &maxCDer)

  The 'goto' constructor: a FSC path is built, starting from a given iSeeML::rob::OrPtConfig and reaching an other one with a maximum curvature and curvature's derivative.

- FscPath (const iSeeML::rob::OrPtConfig &start, const iSeeML::rob::OrPtConfig &goal, const double &maxCurv, const double &maxCDer, const double &limDefl, const double &turnRad, const double &turnAng)

  The complete 'goto' constructor: a FSC path is built, starting from a given iSeeML::rob::OrPtConfig and reaching an other one with a maximum curvature, curvature's derivative, limit deflection, turning radius and turning angle.

- FscPath (const FscPath &other)

  The copy constructor.

- iSeeML::Object & clone () const

  Description method, giving a copy of the current FSC path.

- void writeTo (ostream &O) const

  Description method, writing a description of the path in a given output stream.

- double maxCurvDeriv () const

  Description method, giving the path's maximum curvature's derivative.

- double limDefl () const

  Description method, giving the limit deflection: a turn of lower (but not zero) deflection is degenerated (it is made of two pieces and does not reach maximum curvature derivative nor maximum curvature), a turn of bigger deflection is normal (it is made of three pieces, the middle one being a circular arc of maximum curvature, and the other two being clothoid pieces of maximum curvature derivative).

- double turnRadius () const

  Description method, giving the radius of the turning circle (the circle of all the configurations which can be reached from a fixed one).

- double turnAngle () const

  Description method, giving the constant angle between the turning circle's tangent and the orientation of the configurations which can be reached.

- FscPath & operator= (const FscPath &other)

  Copy operator (default one is not correct).

- void computeCenters (const iSeeML::rob::OrPtConfig &start, const iSeeML::rob::OrPtConfig &goal, iSeeML::geom::Point centers[nbTurningCircles]) const

  Various method, computing the set of circle centers used to turn from starting configuration of the current path, or to turn to reach the given goal configuration.

Static Public Member Functions

- static void computeValues (const double &maxCurv, const double &maxCDer, double &limDefl, double &turnRad, double &turnAng)

  Description method, computing from maximum curvature and maximum curvature's derivative the values of limit deflection, turning radius and turning angle for a FSC path.

Static Public Attributes

- static const string ClassName

  The class name is public, as this class can be instanced.

Protected Member Functions

- bool isShortTurn (const double &defl) const

  Description method, checking whether the given length is too short with respect to the maximum curvature's derivative to reach maximum curvature along a turn.

- int turnNbPieces (const double &defl) const

  Description method, computing the number of pieces needed for a turn of given deflection.

- virtual bool getConnection (const double &dist, double &length, double &angle) const

  Description method, trying to compute the length of the line segment connecting the turning circles and the angle between the segment connecting the turning centers and the previous one.

- void addTurn (int &index, const iSeeML::rob::OrPtConfig **start, const double &defl)

  Modification method, adding to the pieces of a path a turn starting at a given configuration, with a given curvature sign and length.

- iSeeML::rob::DubinsLikePath * connectArray () const

  Various method, returning an array of nbPossiblePaths Dubins-like paths, initialised as clones of the current path.

- iSeeML::rob::DubinsLikePath & getSolution (iSeeML::rob::DubinsLikePath *paths, const int index) const

  Various method, getting in an array of Dubins-like paths the element of given index.

Detailed Description

Objects of this class are Forward-only Sub-optimal Continuous-curvature (FSC) paths, made of circular arcs, pieces of clothoid and line segments. They are similar to Dubins' paths, but their curvature have a continuous profile (pieces of clothoid connecting line segments and circular arcs).

Author:

Alexis Scheuer.

Version:

1.0

Examples:

Fwd.cpp, LengthFwd.cpp, TimeFwd.cpp, and wxGuiFwd.cpp.

Constructor & Destructor Documentation

► **iSeeML::rob::FscPath::FscPath** () **[inline]** The default constructor should only be used for array initializations: it generates a path starting from default oriented point, with type lsl, and zero maximum curvature, maximum curvature's derivative and lengths.

Precondition:

memory should not be full. If memory is full and `ISEEML_CHECK_NIL_POINTER` is defined (see CompilerFlags.h), an error message is generated and the program exits.

See also:

default constructor of iSeeML::rob::DubinsLikePath.

► **iSeeML::rob::FscPath::FscPath (const iSeeML::rob::OrPtConfig & *start*, const Type & *type*, const double & *maxCurv*, const double & *maxCDer*, const double & *length1*, const double & *length2*, const double & *length3*)** **[inline]** The 'forward' constructor: a FSC path is built from its starting oriented point, its type, its maximum curvature, maximum curvature's derivative and parts' lengths.

Parameters:

start the starting configuration,
type the type of Dubins' path,
maxCurv the maximum curvature along the path (taken in absolute value),
maxCDer the maximum curvature's derivative along the path (taken in absolute value),
length1 the length of the first part of the path,
length2 the length of the second part of the path,
length3 the length of the third part of the path.

► **iSeeML::rob::FscPath::FscPath (const iSeeML::rob::OrPtConfig & *start*, const Type & *type*, const double & *maxCurv*, const double & *maxCDer*, const double & *limDefl*, const double & *turnRad*, const double & *turnAng*, const double & *length1*, const double & *length2*, const double & *length3*)** **[inline]** The complete 'forward' constructor: a FSC path is built from its starting oriented point, its type, its maximum curvature, maximum curvature's derivative, limit deflection, turning radius, turning angle, and parts' lengths.

Warning:

this constuctor only works if limit deflection, turning radius and turning angle have been correctly computed (using method computeValues).

Parameters:

start the starting configuration,
type the type of Dubins' path,
maxCurv the maximum curvature along the path (taken in absolute value),
maxCDer the maximum curvature's derivative along the path (taken in absolute value),
limDefl the limit deflection (taken in abs. value),
turnRad the turning radius (taken in abs. value),
turnAng the turning angle (taken in abs. value),
length1 the length of the first part of the path,
length2 the length of the second part of the path,
length3 the length of the third part of the path.

► **iSeeML::rob::FscPath::FscPath (const iSeeML::rob::OrPtConfig & *start*, const iSeeML::rob::OrPtConfig & *goal*, const double & *maxCurv*, const double & *maxCDer*)** `[inline]` The 'goto' constructor: a FSC path is built, starting from a given iSeeML::rob::OrPtConfig and reaching an other one with a maximum curvature and curvature's derivative.

Parameters:

start the starting configuration,
goal the goal configuration,
maxCurv the maximum curvature along the path (taken in absolute value),
maxCDer the maximum curvature's derivative along the path (taken in abs. value).

► **iSeeML::rob::FscPath::FscPath (const iSeeML::rob::OrPtConfig & *start*, const iSeeML::rob::OrPtConfig & *goal*, const double & *maxCurv*, const double & *maxCDer*, const double & *limDefl*, const double & *turnRad*, const double & *turnAng*)** `[inline]` The complete 'goto' constructor: a FSC path is built, starting from a given iSeeML::rob::OrPtConfig and reaching an other one with a maximum curvature, curvature's derivative, limit deflection, turning radius and turning angle.

Warning:

this constuctor only works if limit deflection, turning radius and turning angle have been correctly computed (using the method computeValues).

Parameters:

start the starting configuration,
goal the goal configuration,
maxCurv the maximum curvature along the path (taken in absolute value),

maxCDer the maximum curvature's derivative along the path (taken in abs. value),
limDefl the limit deflection (taken in abs. value),
turnRad the turning radius (taken in abs. value),
turnAng the turning angle (taken in abs. value).

► **iSeeML::rob::FscPath::FscPath (const FscPath & *other*) `[inline]`** The copy constructor.

Parameters:

other the copied path.

Member Function Documentation

► **void iSeeML::rob::FscPath::addTurn (int & *index*, const iSeeML::rob::OrPtConfig ** *start*, const double & *defl*) `[protected, virtual]`** Modification method, adding to the pieces of a path a turn starting at a given configuration, with a given curvature sign and length. Maximum curvature of the current FSC path is used.

Parameters:

index the first free index in the pieces array (pieces will be added at this index and after, and the index will be incremented),
start a reference to the reference of the starting configuration (it will be changed to a ref. to the ref. of next starting config.),
defl the turn's deflection (change of orientation), giving the curvature sign.

Precondition:

This method only works if the maximum curvature is not zero (otherwise, it writes an error on `stderr`).

Implements iSeeML::rob::DubinsLikePath.

► **iSeeML::Object& iSeeML::rob::FscPath::clone () const `[inline, virtual]`** Description method, giving a copy of the current FSC path. This clone is dynamically allocated (using copy constructor), it has to be deleted later.

Returns:

a copy/clone of the current FSC path.

Implements iSeeML::Object.

► **void iSeeML::rob::FscPath::computeCenters (const iSeeML::rob::OrPtConfig & *start*, const iSeeML::rob::OrPtConfig & *goal*, iSeeML::geom::Point *centers*[nbTurningCircles]) const** `[virtual]` Various method, computing the set of circle centers used to turn from starting configuration of the current path, or to turn to reach the given goal configuration.

Parameters:

start the starting configuration,
goal the configuration to reach,
centers the set of circle centers computed.

Precondition:

the maximum curvature should not be zero. If it is, an error message is generated if `ISEEML_CHECK_FSC_PATH_PRECOND` is defined (see CompilerFlags.h).

Implements iSeeML::rob::DubinsLikePath.

► **static void iSeeML::rob::FscPath::computeValues (const double & *maxCurv*, const double & *maxCDer*, double & *limDefl*, double & *turnRad*, double & *turnAng*)** `[static]` Description method, computing from maximum curvature and maximum curvature's derivative the values of limit deflection, turning radius and turning angle for a FSC path. If maximum curvature or curvature's derivative is too small, limit deflection is set to zero and turning radius and angle are given negative values.

Parameters:

maxCurv the maximum curvature along the path (taken in absolute value),
maxCDer the maximum curvature's derivative along the path (taken in abs. value),
limDefl the corresponding limit deflection,
turnRad the corresponding turning radius,
turnAng the corresponding turning angle.

See also:

limDefl, turnRadius, turnAngle.

Examples:

LengthFwd.cpp, and TimeFwd.cpp.

► **iSeeML::rob::DubinsLikePath∗ iSeeML::rob::FscPath::connectArray () const** `[inline, protected, virtual]` Various method, returning an array of nbPossiblePaths Dubins-like paths, initialised as clones of the current path.

Returns:

the array of Dubins-like paths.

Implements iSeeML::rob::DubinsLikePath.

► **virtual bool iSeeML::rob::FscPath::getConnection (const double & *dist*, double & *length*, double & *angle*) const** `[protected, virtual]` Description method, trying to compute the length of the line segment connecting the turning circles and the angle between the segment connecting the turning centers and the previous one. The angle can only be computed if the square distance between the turning centers is greater than a value depending of the path's type. Length is not computed for **lrl** or **rlr** path's type, and is negative when previous angle cannot be computed.

Parameters:

dist the distance between the turning centers,
length the length to compute (modified by the method),
angle the angle to compute (modified by the method).

Returns:

whether the computed was succesful or not (whether the distance `dist2` is greater than a value depending of the path's type).

Implements iSeeML::rob::DubinsLikePath.

► **iSeeML::rob::DubinsLikePath& iSeeML::rob::FscPath::getSolution (iSeeML::rob::DubinsLikePath * *paths*, const int *index*) const** `[inline, protected, virtual]` Various method, getting in an array of Dubins-like paths the element of given index.

Parameters:

paths the array of Dubins-like paths,
index the index of the searched path.

Precondition:

first parameter should be an array of paths obtained using connectArray.
second parameter should be a correct index for first parameter array (between 0 and nbPossiblePaths - 1). If not, an error message is generated if `ISEEML_CHECK_FSC_PATH_PRECOND` is defined, and the number is considered as 1 if smaller and 3 if bigger, if `ISEEML_CHECK_ARRAY_ELEMT` is defined (see CompilerFlags.h).

Returns:

the array of Dubins-like paths.

Implements iSeeML::rob::DubinsLikePath.

► **bool iSeeML::rob::FscPath::isShortTurn (const double & *defl*) const** `[inline, protected]` Description method, checking whether the given length is too short with respect to the maximum curvature's derivative to reach maximum curvature along a turn.

Parameters:

defl the turn's deflection.

Returns:

true if the length is too short, else false.

► **double iSeeML::rob::FscPath::limDefl () const** `[inline]` Description method, giving the limit deflection: a turn of lower (but not zero) deflection is degenerated (it is made of two pieces and does not reach maximum curvature derivative nor maximum curvature), a turn of bigger deflection is normal (it is made of three pieces, the middle one being a circular arc of maximum curvature, and the other two being clothoid pieces of maximum curvature derivative).

Returns:

the turning circle's radius.

Precondition:

This method returns zero if the maximum curvature is zero (zero curvature path should not use it).

► **double iSeeML::rob::FscPath::maxCurvDeriv () const** `[inline]` Description method, giving the path's maximum curvature's derivative.

Returns:

the path's maximum curvature's derivative.

► **FscPath& iSeeML::rob::FscPath::operator= (const FscPath & *other*)** `[inline]` Copy operator (default one is not correct).

Parameters:

other the path to copy.

Returns:

the current path, after modification.

► **double iSeeML::rob::FscPath::turnAngle** () **const** `[inline]` Description method, giving the constant angle between the turning circle's tangent and the orientation of the configurations which can be reached.

Returns:

the turning circle's angle.

Precondition:

This method returns -π if the maximum curvature is zero (zero curvature path should not use it).

See also:

turnRadius.

► **int iSeeML::rob::FscPath::turnNbPieces** (**const double &** ***defl***) **const** `[inline, protected, virtual]` Description method, computing the number of pieces needed for a turn of given deflection.

Parameters:

defl the turn's deflection.

Returns:

the number of needed pieces, which is 1, 2 or 3.

Implements iSeeML::rob::DubinsLikePath.

► **double iSeeML::rob::FscPath::turnRadius** () **const** `[inline, virtual]` Description method, giving the radius of the turning circle (the circle of all the configurations which can be reached from a fixed one).

Returns:

the turning circle's radius.

Precondition:

This method returns -1 if the maximum curvature is zero (zero curvature path should not use it).

Implements iSeeML::rob::DubinsLikePath.

► **void iSeeML::rob::FscPath::writeTo (ostream & *O*) const** `[virtual]` Description method, writing a description of the path in a given output stream. Starting configuration (as an oriented point with curvature) is given, followed by a short description of each piece (curvature's derivative and length), and by the final configuration.

Parameters:

O the output stream in which description is written.

Reimplemented from iSeeML::rob::CompoundPath.

Member Data Documentation

► **const string iSeeML::rob::FscPath::ClassName** `[static]` The class name is public, as this class can be instanced.

See also:

className.

Reimplemented from iSeeML::rob::DubinsLikePath.

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

- include/iSeeML/rob/FscPath.hpp

Chapter 3

Example Documentation

3.0.3.11 Fwd.cpp

This file checks Dubins' and FSC paths generation (their 'goto' constructors): a path of each type is built to connect two given configurations, and its characteristics (including starting and final configurations) are given.

```
//
// This program verifies the correctness
//  of ISeeML' `goto' constructors for forward-only paths
//


// For Borland C++
#ifdef __BORLANDC__
#include <condefs.h>
#endif

// To get the definition of ISeeML' forward paths (Dubins & FSC).
#include <iSeeML/fwdPaths>


int main()
{
  //### Constant definitions ###################################

  // minimum turning radius (clearer than maximum curvature)
  const double minTurnRadius = 5;
  // turning distance (dist. to reach min. turn. rad.)
  const double minTurnDist = 5;


  //### Configurations definitions ############################

  double x, y, th;
  cout << "Enter the starting configuration as x y th "
       << "(in meters and degres): ";
```

```
  cin  >> x >> y >> th;
  iSeeML::rob::OrPtConfig start(x, y, M_PI * th / 180);

  cout << "Enter the final configuration as x y th (same): ";
  cin  >> x >> y >> th;
  iSeeML::rob::OrPtConfig end(x, y, M_PI * th / 180);

  cout << endl;


  //### Dubins' paths computations ##############################

  // maximum curvature is deduced from minimum turning radius
  const double maxCurv = 1 / minTurnRadius;

  // Dubins' path
  iSeeML::rob::DubinsPath DPath(start, end, maxCurv);

  cout << "\nDubins path connecting " << start << " to "
       << end << "\n  is " << DPath << endl;


  //### FSC paths computations ##################################

  // max. curv. derivative is deduced from turning distance
  const double maxCurvDeriv = maxCurv / minTurnDist;

  // Fsc path
  iSeeML::rob::FscPath FPath(start, end, maxCurv, maxCurvDeriv);

  cout << "\nFSC path connecting " << start << " to "
       << end << "\n  is " << FPath << endl << endl;

  return(0);
} // end of main()
```

3.0.3.12 LengthFwd.cpp

This file computes paths' length of Dubins' and FSC paths to connect a random set of configurations' pairs. The mean, standart deviation, minimum and maximum value of the ratio FSC path's length on Dubins' path's length are given, as well as the percentage of ratio under 1.3.

```
//
// This program estimates the length ratio between ISeeML' 'goto'
//  constructors (SFC and Dubins' paths) for forward-only paths
//


// For Borland C++
#ifdef __BORLANDC__
#include <condefs.h>
#endif
```

```
// To get the definition of time
#include <time.h>
// To get the definition of random and srandom
#include <stdlib.h>
// To get the definition of ISeeML' forward paths (Dubins & FSC).
#include <iSeeML/fwdPaths>


int main()
{
  //### Constant definitions ###################################

  // number of configurations (~ sqrt of nb of computations)
  const int nbConfig = 1000;
  // size of the workspace (configurations have (x, y, th)
  //  coordinates with 0 <= x <= xSize, 0 <= y <= ySize and
  //  - pi < th <= pi)
  const double xSize = 50, ySize = 50;
  // minimum turning radius (clearer than maximum curvature)
  const double minTurnRadius = 5;
  // turning distance (dist. to reach min. turn. rad.)
  const double minTurnDist   = 5;


  //### Configurations definitions ############################

  // random seed initialization
#ifdef __BORLANDC__
  srand((unsigned int) time(NULL));
#else
  srandom((unsigned int) time(NULL));
#endif
  // maximum random value
  const int maxRnd = ~(01 << 31);
  // configurations array allocation
  iSeeML::rob::OrPtConfig configs[nbConfig];
  // configuration's initialisations
  int i;
#ifdef __BORLANDC__
#define random rand
#endif
  for(i = 0; i < nbConfig; i++) {
    // coordinates are randomly taken
    const double x = xSize * random() / maxRnd,
      y  = ySize * random() / maxRnd,
      th = (2.0 * random() / maxRnd - 1) * M_PI;
    // configurations is defined
    configs[i] = iSeeML::rob::OrPtConfig(x, y, th); }


  //### Paths computations ####################################

  // variables used in the loop
  int j, nbPaths, nbShort;
  double minRatio, maxRatio, ratioSum, squareSum;
  // maximum curvature is deduced from minimum turning radius
```

```
  const double maxCurv = 1 / minTurnRadius;
  // max. curv. derivative is deduced from turning distance
  const double maxCurvDeriv = maxCurv / minTurnDist;
  // limit defl., turning radius and turning angle
  // are computed (once and for all)
  double limDeflect, turnRadius, turnAngle;
  iSeeML::rob::FscPath::computeValues(maxCurv, maxCurvDeriv,
                                      limDeflect, turnRadius,
                                      turnAngle);

  // computation loop
  for(i = 0, nbPaths = 0, nbShort = 0, minRatio = 2,
        maxRatio = 1, ratioSum = 0, squareSum = 0;
      i < nbConfig; i++)
    for(j = 0; j < nbConfig; j++)
      if (i != j) {
        // compute Dubins' path
        iSeeML::rob::DubinsPath dp(configs[i], configs[j], maxCurv);
        // compute FSC path
        iSeeML::rob::FscPath fscp(configs[i], configs[j], maxCurv,
                                  maxCurvDeriv, limDeflect,
                                  turnRadius, turnAngle);
        // compute the length ratio
        const double ratio = fscp.length() / dp.length();
        // modify the loop variables
        nbPaths++;
        if (ratio <= 1.3)      nbShort++;
        if (ratio < minRatio)  minRatio = ratio;
        if (ratio > maxRatio)  maxRatio = ratio;
        ratioSum  += ratio;
        squareSum += ratio * ratio; }
  // compute final values
  const double mean = ratioSum / nbPaths,
    variance = squareSum / nbPaths - mean * mean;
  // display the result
  cout << "Mean of ratio on " << nbPaths << " computations: "
       << mean << "; std deviation: " << sqrt(variance)
       << endl;
  cout << " minimum: " << minRatio << "; maximum: " << maxRatio
       << "; % under 1.3: " << (double)(nbShort) * 100 / nbPaths
       << endl;

  return(0);
} // end of main()
```

3.0.3.13 TimeFwd.cpp

This file estimates computation time for Dubins' and FSC paths generation (their 'goto' constructors): a set of configurations is randomly generated, and paths connecting any two of these configurations are computed. Global computation time is estimated, and divided by the number of path generations to obtain average computation time of 'goto' constructors for each type of path.

```
//
// This program estimates the computation time
//  of ISeeML' 'goto' constructor for forward-only paths
//


// For Borland C++
#ifdef __BORLANDC__
#include <condefs.h>
#endif

// To get the definition of time
#include <time.h>
// To get the definition of random and srandom
#include <stdlib.h>
// To get the definition of ISeeML' forward paths (Dubins & FSC).
#include <iSeeML/fwdPaths>


int main()
{
  //### Constant definitions ###################################

  // number of configurations (~ sqrt of nb of computations)
  const int nbConfig = 1000;
  // size of the workspace (configurations have (x, y, th)
  //  coordinates with 0 <= x <= xSize, 0 <= y <= ySize and
  //  - pi < th <= pi)
  const double xSize = 50, ySize = 50;
  // minimum turning radius (clearer than maximum curvature)
  const double minTurnRadius = 5;
  // turning distance (dist. to reach min. turn. rad.)
  const double minTurnDist   = 5;


  //### Configurations definitions ############################

  // random seed initialization
#ifdef __BORLANDC__
  srand((unsigned int) time(NULL));
#else
  srandom((unsigned int) time(NULL));
#endif
  // maximum random value
  const int maxRnd = ~(01 << 31);
  // configurations array allocation
  iSeeML::rob::OrPtConfig configs[nbConfig];
  // configuration's initialisations
  int i;
#ifdef __BORLANDC__
#define random rand
#endif
  for(i = 0; i < nbConfig; i++) {
    // coordinates are randomly taken
    const double x = xSize * random() / maxRnd,
      y  = ySize * random() / maxRnd,
      th = (2.0 * random() / maxRnd - 1) * M_PI;
```

```
  // configurations is defined
  configs[i] = iSeeML::rob::OrPtConfig(x, y, th); }


//### Dubins' paths computations ############################

clock_t ctime;
double time, atime;
int j, nbPaths;
// maximum curvature is deduced from minimum turning radius
const double maxCurv = 1 / minTurnRadius;

// starting time
ctime = clock();
// computation loop
for(i = 0, nbPaths = 0; i < nbConfig; i++)
  for(j = 0; j < nbConfig; j++)
    if (i != j) {
      // compute the path (and forget it immediately)
      iSeeML::rob::DubinsPath path(configs[i], configs[j],
                                   maxCurv);
      // increment the paths' counter
      nbPaths++; }
// computation time
ctime = clock() - ctime;
time  = ctime * 1.0 / CLOCKS_PER_SEC;
atime = time * 1000 / nbPaths;
cout << "Elapsed time for " << nbPaths
     << " Dubins' paths computation: \n\t"
     << time << " s, average "<< atime << " ms\n";


//### FSC paths computations ################################

// max. curv. derivative is deduced from turning distance
const double maxCurvDeriv = maxCurv / minTurnDist;
// limit defl., turning radius and turning angle
// are computed (once and for all)
double limDeflect, turnRadius, turnAngle;
iSeeML::rob::FscPath::computeValues(maxCurv, maxCurvDeriv,
                                    limDeflect, turnRadius,
                                    turnAngle);
// starting time
ctime = clock();
// computation loop
for(i = 0, nbPaths = 0; i < nbConfig; i++)
  for(j = 0; j < nbConfig; j++)
    if (i != j) {
      // compute the path (and forget it immediately)
      iSeeML::rob::FscPath path(configs[i], configs[j],
                                maxCurv, maxCurvDeriv,
                                limDeflect, turnRadius,
                                turnAngle);
      // increment the paths' counter
      nbPaths++; }
// computation time
ctime = clock() - ctime;
```

```
  time  = ctime * 1.0 / CLOCKS_PER_SEC;
  atime = time * 1000 / nbPaths;
  cout << "Elapsed time for " << nbPaths
       << " FSC paths computation: \n\t"
       << time << " s, average "<< atime << " ms\n";

  return(0);
} // end of main()
```

3.0.3.14 wxGuiFwd.cpp

This file defines ISeeML v1.0 Graphical User Interface, drawing Dubins' and FSC paths to connect a fixed configuration to a moving one, whose position follows the mouse and whose orientation can be changed using keyboard. This interface is built using (simply) wxWindows (`http://www.wxWindows.org`). Graphical objects' classes will be defined in ISeeML v1.1 to build a GUI toolkit based on wxWindows.

```
//
// This program defines a real-time wxWindows interface drawing
//  ISeeML' forward-only paths connecting two configurations
//


// To get the definition of wxWindows classes
#include "wx/wxprec.h"  // for compilers that support precompilation

#ifdef __BORLANDC__     // addition for Borland C++
    #pragma hdrstop
#endif

#ifndef WX_PRECOMP      // for other compilers
    #include "wx/wx.h"
#endif

// To get the definition of ISeeML' forward paths (Dubins & FSC).
#include <iSeeML/fwdPaths>
// To avoid writing ISeeML::Rob::...
using namespace iSeeML::rob;;


//### wxWindows Drawing Area ###################################

class ISeeMLDrawingArea : public wxPanel {
  wxFrame *parentFrame;
  // Graphic Properties of drawing
  const wxBrush *daGP, *cvGP;   // drawing area and curve fill GP
  const wxPen   *bgGP,    // background pen
    *configGP,            // configurations pen
    *dubinsGP,   *fscpGP, // Dubins' and FSCP paths pens
    *dubinsTCGP, *fscpTCGP;  // turning circles' pens
  wxPoint centerPoint;    // middle point of drawing area
  double  scale;          // drawing scale
  static const OrPtConfig start;  // path(s) starting config.
```

```
  OrPtConfig goal;        // goal config
  bool    movingConfig;   // does goal config follow mouse
  double  maxCurv, maxDCurv;  // max. curvature and derivative
  DubinsPath dubins;      // Dubins' path
  FscPath fscp;           // forward continuous-curvature path
public:
  // === Constructor ============================================
  ISeeMLDrawingArea(wxFrame& parent, double scl) :
    wxPanel(&parent),         parentFrame(&parent),
    scale(scl),               movingConfig(false),
    maxCurv(1),               maxDCurv(0.8),
    dubins(start, goal, maxCurv),
    fscp(start, goal, maxCurv, maxDCurv)  {
    daGP       = wxWHITE_BRUSH;
    cvGP       = wxTRANSPARENT_BRUSH;
    bgGP       = wxWHITE_PEN;
    configGP   = wxRED_PEN;
    dubinsGP   = new wxPen( wxT("BLUE") );
    fscpGP     = wxGREEN_PEN;
    dubinsTCGP = new wxPen( wxT("ORANGE"), 1, wxLONG_DASH );
    fscpTCGP   = new wxPen( wxT("GOLD"),   1, wxLONG_DASH );
    SetBackgroundColour(*wxWHITE);
  }

  // === Event Handlers =========================================
  void OnMouseClick(wxMouseEvent& event);
  void OnMouseMove (wxMouseEvent& event);
  void OnMouseWheel(wxMouseEvent& event);
  void OnSize      (wxSizeEvent&  event);
  void OnPaint     (wxPaintEvent& event);
  void OnChar      (wxKeyEvent&   event);

private:
  // Conversion from real points to pixel points
  wxPoint dblPt2intPt(const iSeeML::geom::Point& p) const {
    return wxPoint( centerPoint.x + (int)(p.xCoord() * scale),
                    centerPoint.y + (int)(p.yCoord() * scale) );
  }

  // === Drawing Handlers =======================================
  void Draw  (wxDC& dc, const OrPtConfig& q);
  void Draw  (wxDC& dc, const DubinsLikePath& P);
  void DrawTC(wxDC& dc, const DubinsLikePath& P);
  void Draw  (wxDC& dc, const bool erase = false);
  void ChangeGoal(const OrPtConfig& q);

  DECLARE_EVENT_TABLE()
}; // end of class ISeeMLDrawingArea


//### wxWindows Window ########################################

class ISeeMLFrame : public wxFrame
{
  enum { ID_MENU_EDIT_CONFIG = 1,   ID_MENU_EDIT_MAXIMA,
         ID_MENU_EDIT_ZOOM,         ID_MENU_VIEW_PATH,
         ID_MENU_VIEW_PATH_DUBINS,  ID_MENU_VIEW_PATH_FSCP,
```

```
        ID_MENU_VIEW_PATH_BOTH,    ID_MENU_VIEW_TURN_CIRCLES
  };

  wxMenu*                      pathSubMenu;
  ISeeMLDrawingArea* drawingArea;

public:
  // === Constructor ================================================
  ISeeMLFrame(const wxString& title);

  // === Descriptions ===============================================
  bool DrawDubins() const {
    return ( pathSubMenu->IsChecked(ID_MENU_VIEW_PATH_DUBINS) ||
             pathSubMenu->IsChecked(ID_MENU_VIEW_PATH_BOTH)   );}
  bool DrawFSCP() const {
    return ( pathSubMenu->IsChecked(ID_MENU_VIEW_PATH_FSCP) ||
             pathSubMenu->IsChecked(ID_MENU_VIEW_PATH_BOTH) ); }
  bool ShowTurnCircles() const {
    wxMenu *editMenu = GetMenuBar()->GetMenu(2);
    return editMenu->IsChecked(ID_MENU_VIEW_TURN_CIRCLES); }

  // === Modifications ==============================================
  void enableSave(bool enable)
  { GetMenuBar()->GetMenu(0)->Enable(wxID_SAVEAS, enable); }

  // === Event Handlers =============================================
  // --- File Menu's Events -----------------------------------------
  void OnSaveAs(wxCommandEvent& event);
  void OnQuit  (wxCommandEvent& WXUNUSED(event)) { Close(TRUE); }
  // --- Edit Menu's Events -----------------------------------------
  void OnSetGoal     (wxCommandEvent& event);
  void OnSetMaxima   (wxCommandEvent& event);
  void OnSetZoom     (wxCommandEvent& event);
  // --- View Menu's Events -----------------------------------------
  void OnSelectPath  (wxCommandEvent& event);
  void OnShowTurnCirc(wxCommandEvent& WXUNUSED(event))
  { drawingArea->Refresh(); }
  // --- Help Menu's Events -----------------------------------------
  void OnAbout       (wxCommandEvent& event);

private:
  DECLARE_EVENT_TABLE()
}; // end of class ISeeMLFrame


//### wxWindows application ###################################

class ISeeMLApp : public wxApp
{
  bool OnInit()
  {
    ISeeMLFrame *frame =
      new ISeeMLFrame( wxT("ISeeML V1.0 Interface") );
    frame->Show(TRUE);
    SetTopWindow(frame);
    return TRUE;
  } // end of bool OnInit()
```

```
}; // end of class ISeeMLApp

IMPLEMENT_APP(ISeeMLApp)


//### wxWindows Drawing Area ###################################

// === Static Field ============================================

const OrPtConfig ISeeMLDrawingArea::start
= OrPtConfig();

// === Event Handlers ==========================================

BEGIN_EVENT_TABLE(ISeeMLDrawingArea, wxPanel)
  EVT_LEFT_DOWN  (ISeeMLDrawingArea::OnMouseClick)
  EVT_MOTION     (ISeeMLDrawingArea::OnMouseMove)
  EVT_MOUSEWHEEL (ISeeMLDrawingArea::OnMouseWheel)
  EVT_SIZE       (ISeeMLDrawingArea::OnSize)
  EVT_PAINT      (ISeeMLDrawingArea::OnPaint)
  EVT_KEY_DOWN   (ISeeMLDrawingArea::OnChar)
END_EVENT_TABLE()

// --- Mouse Click Handler -------------------------------------
void ISeeMLDrawingArea::OnMouseClick(wxMouseEvent& event)
{
  movingConfig = !movingConfig;
  OnMouseMove(event);

  wxString message;
  if (movingConfig) {
    ((ISeeMLFrame *) parentFrame)->enableSave(false);
    message.Printf(wxT("Clic to fix, use arrows or")
                   wxT(" page up/down to change orientation")
                   ); }
  else {
    ((ISeeMLFrame *) parentFrame)->enableSave(true);
    message.Printf(wxT("Clic to change again")
                   wxT(" goal configuration")
                   ); }
  parentFrame->SetStatusText(message);
} // end of ISeeMLDrawingArea::OnMouseClick(...)

// --- Mouse Motion Handler ------------------------------------
void ISeeMLDrawingArea::OnMouseMove(wxMouseEvent &event)
{
  if (movingConfig) {
    // prepare the device context associated to the drawing area
    wxClientDC dc(this);
    PrepareDC(dc);
    parentFrame->PrepareDC(dc);

    // get mouse coordinates in this device context
    wxPoint pos = event.GetPosition();
    int i = (int) dc.DeviceToLogicalX(pos.x),
        j = (int) dc.DeviceToLogicalY(pos.y);
```

```
    // change goal configuration and redraw
    OrPtConfig newGoal( (i - centerPoint.x) / scale,
                        (j - centerPoint.y) / scale,
                       goal.orientation() );
    ChangeGoal(newGoal);
  } // end of if (movingConfig)
} // end of ISeeMLDrawingArea::OnMouseMove(...)

// --- Mouse's Wheel Motion Handler -----------------------------
void ISeeMLDrawingArea::OnMouseWheel(wxMouseEvent &event)
{
  static const double PI_196 = M_PI / 196, PI_28 = M_PI / 28,
    PI_4 = M_PI / 4;
  // motion of the wheel, in +/- 1
  const int change =  goal.sign( event.GetWheelRotation() );
  // using event.GetWheelDelta() does not work (it returns 0)
  double newTh = goal.orientation();

  if ( event.ShiftDown() )          // SHIFT => big changes
    newTh += change * PI_4;
  else if ( event.MiddleIsDown() ) // wheel pressed & not SHIFT
    newTh += change * PI_196;       // => small changes
  else newTh += change * PI_28;     // else normal changes

  // change goal configuration and redraw
  OrPtConfig newGoal(goal.position(), newTh);
  ChangeGoal(newGoal);
} // end of ISeeMLDrawingArea::OnMouseWheel(...)

// --- Window's Resize Handler ----------------------------------
void ISeeMLDrawingArea::OnSize(wxSizeEvent &event)
{
  wxSize size   = event.GetSize();
  centerPoint.x = size.GetWidth()  / 2;
  centerPoint.y = size.GetHeight() / 2;
  Refresh();
} // end of ISeeMLDrawingArea::OnSize(wxSizeEvent&)

// --- Window's Refresh Handler ---------------------------------
void ISeeMLDrawingArea::OnPaint(wxPaintEvent &event)
{
  // prepare the device context associated to the drawing area
  wxClientDC dc(this);
  PrepareDC(dc);
  parentFrame->PrepareDC(dc);

  // redraw background
  dc.StartPage(); // replace BeginDrawing() ?
  ClearBackground();
  // redraw path(s) and configuations
  dc.SetBrush(*cvGP);
  Draw(dc);
  dc.EndPage(); // replace EndDrawing() ?
} // end of ISeeMLDrawingArea::OnPaint(wxPaintEvent&)
```

```
// --- Keyboard Handler -----------------------------------------
void ISeeMLDrawingArea::OnChar(wxKeyEvent &event)
{
  static const double PI_196 = M_PI / 196, PI_28 = M_PI / 28,
    PI_4 = M_PI / 4;
  double newTh = goal.orientation();
  if (movingConfig)
    switch ( event.GetKeyCode() ) {
    case WXK_HOME : newTh = 0;        break;
    case WXK_END  : newTh = M_PI;     break;
    case WXK_NEXT : newTh += PI_4;    break;
    case WXK_PRIOR: newTh -= PI_4;    break;
    case WXK_DOWN : newTh += PI_28;   break;
    case WXK_UP   : newTh -= PI_28;   break;
    case WXK_RIGHT: newTh += PI_196;  break;
    case WXK_LEFT : newTh -= PI_196;  break;
    default       : event.Skip();
    } // end of switch
  else
    event.Skip();

  // change goal configuration and redraw
  OrPtConfig newGoal(goal.position(), newTh);
  ChangeGoal(newGoal);
} // end of ISeeMLDrawingArea::OnChar(wxKeyEvent&)


// === Drawing Handlers ========================================

// --- Configuration Drawing ------------------------------------
void ISeeMLDrawingArea::Draw(wxDC& dc, const OrPtConfig& q)
{
  // coordinates computation
  const static int configSize = 20;
  wxPoint startP = dblPt2intPt( q.position() );
  int di = (int)(configSize * cos( q.orientation() )),
      dj = (int)(configSize * sin( q.orientation() )),
      iF = startP.x + di, jF = startP.y + dj;
  // drawing
  dc.DrawLine(startP.x, startP.y, iF, jF);
  dc.DrawLine(iF, jF, iF - (di + dj) / 4, jF + (di - dj) / 4);
  dc.DrawLine(iF, jF, iF - (di - dj) / 4, jF - (di + dj) / 4);
} // end of ISeeMLDrawingArea::Draw(... OrPtConfig&)

// --- Path Drawing ---------------------------------------------
void ISeeMLDrawingArea::Draw(wxDC& dc, const DubinsLikePath& P)
{
  int i = 0;
  wxPoint startP = dblPt2intPt( P.start().position() );
  while( i < P.nbPieces() ) {
    const LinCurvPath& linPiece = P.lcPiece(++i);
    wxPoint endP = dblPt2intPt( linPiece.end().position() );
    // short circular arcs are drawn as complete circles,
    // it is easily avoided (no drawing when piece's length
    // is smaller than a pixel's maximum size = diagonal)
    if (linPiece.length() > 1.5 / scale)
      if ( P.isZero( linPiece.sharpness() ) )
```

```
        // curvature does not change along piece
        if ( P.isZero( linPiece.start().curvature() ) )
          // piece = line segment
          dc.DrawLine(startP.x, startP.y, endP.x, endP.y);
        else {
          // piece = circular arc
          iSeeML::geom::Vector v( linPiece.start().orientation() );
          wxPoint cntr = dblPt2intPt
            ( linPiece.start().position() +
              v.rotate(M_PI_2) / linPiece.start().curvature() );
          if (linPiece.start().curvature() > 0)
            dc.DrawArc(endP.x, endP.y, startP.x, startP.y,
                       cntr.x, cntr.y);
          else
            dc.DrawArc(startP.x, startP.y, endP.x, endP.y,
                       cntr.x, cntr.y); } // enf of else (circ arc)
      else {
        // piece = piece of clothoid
        double step = 5. / scale, arcLngth = step;
        while (arcLngth < linPiece.length()) {
          wxPoint midP =
            dblPt2intPt( linPiece[arcLngth].position() );
          dc.DrawLine(startP.x, startP.y, midP.x, midP.y);
          startP = midP;
          arcLngth += step; }
        dc.DrawLine(startP.x, startP.y, endP.x, endP.y); }
    else
      dc.DrawLine(startP.x, startP.y, endP.x, endP.y);
    startP = endP;
  } // end of while
} // end of ISeeMLDrawingArea::Draw(...Path&)

// --- Path's Turning Circles Drawing ----------------------------
void ISeeMLDrawingArea::DrawTC(wxDC& dc, const DubinsLikePath& P)
{
  // compute turning circles' centers
  iSeeML::geom::Point centers[P.nbTurningCircles];
  P.computeCenters(P.start(), P.end(), centers);

  int i, radius = (int) (P.turnRadius() * scale);
  for(i = 0; i < P.nbTurningCircles; i++)
    dc.DrawCircle(dblPt2intPt(centers[i]), radius);
} // end of ISeeMLDrawingArea::DrawTC(...Path&)

// --- Drawing / Erasing Everything ------------------------------
void ISeeMLDrawingArea::Draw(wxDC& dc, const bool clean)
{
  // graphical properties setting
  if (clean)    dc.SetPen(*bgGP);
  else          dc.SetPen(*configGP);
  dc.SetBrush(*cvGP);
  // drawing starting and goal configurations
  Draw(dc, start);
  Draw(dc, goal);
  ISeeMLFrame *parent = (ISeeMLFrame *) parentFrame;
  // drawing paths
  if ( clean || parent->DrawDubins() ) {
```

```
    if (!clean) dc.SetPen(*dubinsGP);
    Draw(dc, dubins);
    if ( clean || parent->ShowTurnCircles() ) {
      if (!clean) dc.SetPen(*dubinsTCGP);
      DrawTC(dc, dubins); } }
  if ( clean || parent->DrawFSCP() ) {
    if (!clean) dc.SetPen(*fscpGP);
    Draw(dc, fscp);
    if ( clean || parent->ShowTurnCircles() ) {
      if (!clean) dc.SetPen(*fscpTCGP);
      DrawTC(dc, fscp); } }
} // end of ISeeMLDrawingArea::Draw(..., const wxPen&)

// --- Goal Config. Change ------------------------------------
void ISeeMLDrawingArea::ChangeGoal(const OrPtConfig& q)
{
  // prepare the device context associated to the drawing area
  wxClientDC dc(this);
  PrepareDC(dc);
  parentFrame->PrepareDC(dc);

  dc.StartPage(); // replace BeginDrawing() ?
  // erase previous path(s) (faster than dc.clear()!)
  Draw(dc, true);
  // change goal config. and paths
  goal   = q;
  dubins.connect(start, goal);
  fscp.connect(start, goal);
  // redraw new path(s)
  Draw(dc);
  dc.EndPage(); // replace EndDrawing() ?
} // end of ISeeMLDrawingArea::ChangeGoal(...)


//### wxWindows Window ########################################

// === Constructor ============================================

ISeeMLFrame::ISeeMLFrame(const wxString& title)
  : wxFrame((wxFrame *) NULL, -1, title, wxDefaultPosition,
            wxSize(750, 800))
{
  // Shortcuts used in menus :
  //   A, B, C, D, L, M, S, T, X, Z

  wxMenuBar *menuBar = new wxMenuBar;

  wxMenu    *menu    = new wxMenu;
  menu->Append( wxID_SAVEAS, wxT("&Save/Export as...\tS"),
                wxT("Save drawing into various formats") );
  menu->AppendSeparator();
  menu->Append( wxID_EXIT,   wxT("E&xit\tX"),
                wxT("Close GUI and quit") );
  menuBar->Append( menu, wxT("File") );
```

```
  menu = new wxMenu;
  menu->Append( ID_MENU_EDIT_CONFIG,
                wxT("Set Goal &Configuration...\tC"),
                wxT("Not yet available!") );
  menu->AppendSeparator();
  menu->Append( ID_MENU_EDIT_MAXIMA,
                wxT("Modify Paths' &Maxima...\tM"),
                wxT("Not yet available!") );
  menu->AppendSeparator();
  menu->Append( ID_MENU_EDIT_ZOOM, wxT("Change &Zoom/Scale...\tZ"),
                wxT("Not yet available!") );
  menuBar->Append( menu, wxT("Edit") );

  menu        = new wxMenu;
  pathSubMenu = new wxMenu;
  pathSubMenu->AppendCheckItem( ID_MENU_VIEW_PATH_DUBINS,
                                wxT("&Dubins'\tD"),
                                wxT("Only draw Dubin's path") );
  pathSubMenu->AppendCheckItem( ID_MENU_VIEW_PATH_FSCP,
                                wxT("&Linear Curvature (FSCP)\tL"),
                                wxT("Only draw FSC path") );
  pathSubMenu->AppendCheckItem( ID_MENU_VIEW_PATH_BOTH,
                                wxT("&Both\tB"),
                                wxT("Draw both Dubin's and FSC paths") );
  pathSubMenu->Check(ID_MENU_VIEW_PATH_DUBINS, true);
  menu->Append(ID_MENU_VIEW_PATH, wxT("Draw Paths"), pathSubMenu);
  menu->AppendCheckItem( ID_MENU_VIEW_TURN_CIRCLES,
                         wxT("Show &Turning Circles...\tT"),
                         wxT("Display turning circles")
                         wxT(" of drawn path(s)") );
  menuBar->Append( menu, wxT("View") );

  menu = new wxMenu;
  menu->Append( wxID_ABOUT, wxT("&About...\tA"),
                wxT("Display information about this application") );
  menuBar->Append( menu, wxT("Help") );

  SetMenuBar(menuBar);

  CreateStatusBar();
  wxString message;
  message.Printf(wxT("Clic in the drawing array to change")
                 wxT(" goal configuration")
                 );
  SetStatusText(message);

  drawingArea = new ISeeMLDrawingArea(*this, 50);
} // end of ISeeMLFrame(const wxString&, ...)


// === Event Handlers ============================================

BEGIN_EVENT_TABLE(ISeeMLFrame, wxFrame)
  // --- File Menu ---
  EVT_MENU (wxID_SAVEAS, ISeeMLFrame::OnSaveAs)
  EVT_MENU (wxID_EXIT,   ISeeMLFrame::OnQuit)
```

```
  // --- Edit Menu ---
  EVT_MENU (ID_MENU_EDIT_CONFIG,
            ISeeMLFrame::OnSetGoal)
  EVT_MENU (ID_MENU_EDIT_MAXIMA,
            ISeeMLFrame::OnSetMaxima)
  EVT_MENU (ID_MENU_EDIT_ZOOM,
            ISeeMLFrame::OnSetZoom)
  // --- View Menu ---
  EVT_MENU (ID_MENU_VIEW_PATH_DUBINS,
            ISeeMLFrame::OnSelectPath)
  EVT_MENU (ID_MENU_VIEW_PATH_FSCP,
            ISeeMLFrame::OnSelectPath)
  EVT_MENU (ID_MENU_VIEW_PATH_BOTH,
            ISeeMLFrame::OnSelectPath)
  EVT_MENU (ID_MENU_VIEW_TURN_CIRCLES,
            ISeeMLFrame::OnShowTurnCirc)
  // --- Help Menu ---
  EVT_MENU (wxID_ABOUT, ISeeMLFrame::OnAbout)
END_EVENT_TABLE()

// --- "Save/Export as..." MenuItem Handler ---------------------
void ISeeMLFrame::OnSaveAs(wxCommandEvent& WXUNUSED(event))
{
  SetStatusText( wxT("Sorry, exporting drawing ")
                 wxT("is not yet implemented!") );
} // end of ISeeMLFrame::OnSaveAs(wxCommandEvent&)

// --- "Set Goal Configuration..." MenuItem Handler -------------
void ISeeMLFrame::OnSetGoal(wxCommandEvent& WXUNUSED(event))
{
  SetStatusText( wxT("Sorry, goal definition's dialog box ")
                 wxT("is not yet implemented!") );
} // end of ISeeMLFrame::OnSetGoal(wxCommandEvent&)

// --- "Modify Paths' Maxima" MenuItem Handler ------------------
void ISeeMLFrame::OnSetMaxima(wxCommandEvent& WXUNUSED(event))
{
  SetStatusText( wxT("Sorry, maxima changing is not yet implemented!") );
} // end of ISeeMLFrame::OnSetMaxima(wxCommandEvent&)

// --- "Change Zoom/Scale" MenuItem Handler ---------------------
void ISeeMLFrame::OnSetZoom(wxCommandEvent& WXUNUSED(event))
{
  SetStatusText( wxT("Sorry, zoom/scale changing ")
                 wxT("is not yet implemented!") );
} // end of ISeeMLFrame::OnSetZoom(wxCommandEvent&)

// --- "Draw Paths" MenuItem Handler ----------------------------
void ISeeMLFrame::OnSelectPath(wxCommandEvent& event)
{
  pathSubMenu->Check(ID_MENU_VIEW_PATH_DUBINS, false);
  pathSubMenu->Check(ID_MENU_VIEW_PATH_FSCP,   false);
  pathSubMenu->Check(ID_MENU_VIEW_PATH_BOTH,   false);
  pathSubMenu->Check(event.GetId(),            true);

  drawingArea->Refresh();
} // end of ISeeMLFrame::OnSelectPath(wxCommandEvent&)
```

```
// --- "About..." MenuItem Handler -----------------------------
void ISeeMLFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
  wxString message;
  message.Printf(wxT("This is ISeeML V1.0 GUI\n\n")
                 wxT("It uses wxWidgets (www.wxWidgets.org),\n")
                 wxT("which should become\n")
                 wxT("ISeeML V1.1 base GUI toolkit.\n\n")
                 wxT("Written by A. Scheuer\n")
                 wxT("(Alexis.Scheuer@inria.fr)")
                 );

  wxMessageBox(message, _T("About ISeeML V1.0 Interface"),
               wxOK | wxICON_INFORMATION, this);
} // end of ISeeMLFrame::OnAbout(wxCommandEvent&)
```

Index

Bibliography

[1] J.-D. Boissonnat, A. Cerezo, and J. Leblond. Shortest paths of bounded curvature in the plane. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 2315–2320, Nice (FR), May 1992. Complete description of these results in the INRIA research report 1503, June 1991.

[2] X.-N. Bui, P. Souères, J.-D. Boissonnat, and J.-P. Laumond. Shortest path synthesis for Dubins non-holonomic robot. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 2–7, San Diego, CA (US), May 1994. `http://dbserver.laas.fr/pls/LAAS/publis.rech_doc?langage=fr&clef=9245`.

[3] H. Cecotti. Planification de chemin utilisant des méthodes analytiques et stochastiques. Mémoire de Diplôme d'Études Approfondies, Loria / Univ. Henri Poincaré, Nancy (FR), July 2002.

[4] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

[5] Th. Fraichard and J.-M. Ahuactzin. Smooth path planning for cars. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Seoul (KR), May 2001. `http://emotion.inrialpes.fr/fraichard/publications`.

[6] Th. Fraichard, A. Scheuer, and R. Desvigne. From Reeds and Shepp's to continuous-curvature paths. In *Proc. of the IEEE Int. Conf. on Advanced Robotics*, pages 585–590, Tokyo (JP), October 1999. `http://www.loria.fr/~scheuer/publis.html#p99c`.

[7] A. Scheuer and Th. Fraichard. Continuous-curvature path planning for car-like vehicles. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 2, pages 997–1003, Grenoble (FR), September 1997. `http://www.loria.fr/~scheuer/publis.html#p97b`.

[8] A. Scheuer and Ch. Laugier. Planning sub-optimal and continuous-curvature paths for car-like robots. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 1, pages 25–31, Victoria, BC (CA), October 1998. `http://www.loria.fr/~scheuer/publis.html#p98`.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803