DCA

Generated by Doxygen 1.9.1

Chapter 1

DCA - Differentiable Collision Avoidance

We implement differentiable collision avoidance using multiple shape primitives.

1.1 Primitives

The library supports the following primitives with the mentioned states and parameters:

- Sphere, where the state is the center of the sphere.
- Capsule, where the state is the two ends of the capsule, stacked.
- Rectangle, where the state is the center of mass and the orientation (expressed in exponential coordinates).
- Box, where the state is the center of mass and the orientation (expressed in exponential coordinates).

Furthermore, the size of the parameters is:

- 0 for a sphere
- 1 for a capsule
- 2 for a rectangle
- 3 for a box

1.2 Example

We refer the reader to the `example`.

1.3 API

The API can be found inside the `API header`. Generally speaking, one passes two primitives and gets back either the distance or the first or second derivative. If the states of the two primitives do not change, one can also pass the computed parameterization (`t`) using the specialized functions. The parameterization can also be computed with a single API call and the two given primitives.

1.4 Documentation

The documentation can be downloaded `here`.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Module Documentation

7.1 Partial Derivatives of Rotation Matrices

Second derivatives of $R(\theta)$.

Functions

- Eigen::Matrix3d DCA::ddR_0_0 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_0 d\theta_0}$.
- Eigen::Matrix3d DCA::ddR_0_1 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_0 d\theta_1}$.
- Eigen::Matrix3d DCA::ddR_0_2 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_0 d\theta_2}$.
- Eigen::Matrix3d DCA::ddR_1_0 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_1 d\theta_0}$.
- Eigen::Matrix3d DCA::ddR_1_1 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_1 d\theta_1}$.
- Eigen::Matrix3d DCA::ddR_1_2 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_1 d\theta_2}$.
- Eigen::Matrix3d DCA::ddR_2_0 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_2 d\theta_0}$.
- Eigen::Matrix3d DCA::ddR_2_1 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_2 d\theta_1}$.
- Eigen::Matrix3d DCA::ddR_2_2 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_2 d\theta_2}$.

7.1.1 Detailed Description

Second derivatives of $R(\theta)$.

θ represents the exponential map parameter vector of the rotation.

$\frac{dR}{d\theta_i}$ is the derivative of the rotation matrix with respect to the i-th exponential map parameter $\frac{d^2R}{d\theta_i d\theta_j}$ is the derivative of $\frac{dR}{d\theta_i}$ with respect to the j-th exponential map parameter

7.1.2 Function Documentation

7.1.2.1 ddR_0_0()

```
Eigen::Matrix3d DCA::ddR_0_0 (
            const Eigen::Vector3d & v )  [inline]
```

Compute $\frac{d^2R}{d\theta_0 d\theta_0}$.

Parameters

in	*v*	θ

Returns

The partial second derivative.

7.1.2.2 ddR_0_1()

```
Eigen::Matrix3d DCA::ddR_0_1 (
            const Eigen::Vector3d & v )  [inline]
```

Compute $\frac{d^2R}{d\theta_0 d\theta_1}$.

Parameters

in	*v*	θ

Returns

The partial second derivative.

7.1.2.3 ddR_0_2()

```
Eigen::Matrix3d DCA::ddR_0_2 (
            const Eigen::Vector3d & v )  [inline]
```

Compute $\frac{d^2R}{d\theta_0 d\theta_2}$.

Parameters

in	*v*	θ

Returns

The partial second derivative.

7.1.2.4 ddR_1_0()

```
Eigen::Matrix3d DCA::ddR_1_0 (
            const Eigen::Vector3d & v )  [inline]
```

Compute $\frac{d^2R}{d\theta_1 d\theta_0}$.

Parameters

in	*v*	θ

Returns

The partial second derivative.

7.1.2.5 ddR_1_1()

```
Eigen::Matrix3d DCA::ddR_1_1 (
            const Eigen::Vector3d & v )  [inline]
```

Compute $\frac{d^2R}{d\theta_1 d\theta_1}$.

Parameters

in	*v*	θ

Returns

The partial second derivative.

7.1.2.6 ddR_1_2()

```
Eigen::Matrix3d DCA::ddR_1_2 (
            const Eigen::Vector3d & v )  [inline]
```

Compute $\frac{d^2 R}{d\theta_1 d\theta_2}$.

Parameters

in	*v*	θ

Returns

The partial second derivative.

7.1.2.7 ddR_2_0()

```
Eigen::Matrix3d DCA::ddR_2_0 (
            const Eigen::Vector3d & v )  [inline]
```

Compute $\frac{d^2R}{d\theta_2 d\theta_0}$.

Parameters

in	*v*	θ

Returns

The partial second derivative.

7.1.2.8 ddR_2_1()

```
Eigen::Matrix3d DCA::ddR_2_1 (
            const Eigen::Vector3d & v )  [inline]
```

Compute $\frac{d^2R}{d\theta_2 d\theta_1}$.

Parameters

in	*v*	θ

Returns

The partial second derivative.

7.1.2.9 ddR_2_2()

```
Eigen::Matrix3d DCA::ddR_2_2 (
            const Eigen::Vector3d & v )  [inline]
```

Compute $\frac{d^2 R}{d\theta_2 d\theta_2}$.

Parameters

in	v	θ

Returns

The partial second derivative.

Chapter 8

Namespace Documentation

8.1 DCA Namespace Reference

Classes

- class API

  Public API.
- class ExpCoords

  Represents Rotations using exponential coordinates.
- class FiniteDifference

  Helper for finite differences (to test derivatives).
- class Logger

  Logger class.
- class NewtonOptimizer

  Simply newton optimizer class.
- class Objective

  The main objective which used for computing distances and their derivatives.
- class PermutationPairGenerator

  This generator creates all possible permutations of pairs. This means, the amount of pairs created is $n^2/2$, where n is the number of primitives. Pairs are not returned twice (0, 1) and (1, 0).
- class NeighborsPairGenerator

  This generator computes the pairs which are in a certain threshold from each other. It does so by computing a single position for each primitive and selecting pairs based on the distance.
- class Primitive

  Definition of a Primitive.
- class Sphere

  Definition of a Sphere.
- class Capsule

  Definition of a Capsule.
- class Rectangle

  Definition of a Rectangle.
- class Box

  Definition of a Box.
- class SoftUnilateralConstraint

  This class is a soft constraint, meaning a constraint is softified.
- class Solver

  Wrapper around a newton optimizer and an objective.

Typedefs

- using primitive_t = std::variant< Sphere, Capsule, Rectangle, Box >

  All possible primitives.
- typedef unsigned int uint

  Easier access to a unsigned int.
- using Vector0d = Eigen::Matrix< double, 0, 1 >

  Easier access to a 0 vector of type double.
- using Vector1d = Eigen::Matrix< double, 1, 1 >

  Easier access to a 1 vector of type double.
- using Vector6d = Eigen::Matrix< double, 6, 1 >

  Easier access to a 6 vector of type double.
- using Vector12d = Eigen::Matrix< double, 12, 1 >

  Easier access to a 12 vector.
- using Matrix12d = Eigen::Matrix< double, 12, 12 >

  Easier access to a 12 by 12 matrix.
- using pair_t = std::pair< size_t, size_t >

  Easier access to a pair (corresponding of two indices)

Functions

- Eigen::Matrix3d ddR_0_0 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_0 d\theta_0}$.
- Eigen::Matrix3d ddR_0_1 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_0 d\theta_1}$.
- Eigen::Matrix3d ddR_0_2 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_0 d\theta_2}$.
- Eigen::Matrix3d ddR_1_0 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_1 d\theta_0}$.
- Eigen::Matrix3d ddR_1_1 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_1 d\theta_1}$.
- Eigen::Matrix3d ddR_1_2 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_1 d\theta_2}$.
- Eigen::Matrix3d ddR_2_0 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_2 d\theta_0}$.
- Eigen::Matrix3d ddR_2_1 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_2 d\theta_1}$.
- Eigen::Matrix3d ddR_2_2 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_2 d\theta_2}$.

8.1.1 Typedef Documentation

8.1.1.1 primitive_t

```
using DCA::primitive_t = typedef std::variant<Sphere, Capsule, Rectangle, Box>
```

All possible primitives.

8.1.1.2 uint

```
typedef unsigned int DCA::uint
```

Easier access to a unsigned int.

8.1.1.3 Vector0d

```
using DCA::Vector0d = typedef Eigen::Matrix<double, 0, 1>
```

Easier access to a 0 vector of type double.

8.1.1.4 Vector1d

```
using DCA::Vector1d = typedef Eigen::Matrix<double, 1, 1>
```

Easier access to a 1 vector of type double.

8.1.1.5 Vector6d

```
using DCA::Vector6d = typedef Eigen::Matrix<double, 6, 1>
```

Easier access to a 6 vector of type double.

8.1.1.6 Vector12d

```
using DCA::Vector12d = typedef Eigen::Matrix<double, 12, 1>
```

Easier access to a 12 vector.

8.1.1.7 Matrix12d

```
using DCA::Matrix12d = typedef Eigen::Matrix<double, 12, 12>
```

Easier access to a 12 by 12 matrix.

8.1.1.8 pair_t

```
using DCA::pair_t = typedef std::pair<size_t, size_t>
```

Easier access to a pair (corresponding of two indices)

Chapter 9

Class Documentation

9.1 DCA::API Class Reference

Public API.

`#include <API.h>`

Static Public Member Functions

- static double compute_D (const primitive_t &p_a, const primitive_t &p_b)

    Computes the shortest distance between two primitives.
- static void compute_dDdS (Vector12d &dDdS, const primitive_t &p_a, const primitive_t &p_b)

    Computes the first derivative of the shortest distance.
- static void compute_d2DdS2 (Matrix12d &d2DdS2, const primitive_t &p_a, const primitive_t &p_b)

    Computes the second derivative of the shortest distance.
- static std::pair$<$ Vector3d, Vector3d $>$ compute_closest_points (const primitive_t &p_a, const primitive_t &p_b)

    Compute the points which yield the shortest distance.
- static double compute_D (const primitive_t &p_a, const primitive_t &p_b, const VectorXd &t)

    Computes the shortest distance between two primitives given t.
- static void compute_dDdS (Vector12d &dDdS, const primitive_t &p_a, const primitive_t &p_b, const VectorXd &t)

    Computes the first derivative of the shortest distance given t.
- static void compute_d2DdS2 (Matrix12d &d2DdS2, const primitive_t &p_a, const primitive_t &p_b, const VectorXd &t)

    Computes the second derivative of the shortest distance given t.
- static std::pair$<$ Vector3d, Vector3d $>$ compute_closest_points (const primitive_t &p_a, const primitive_t &p_b, const VectorXd &t)

    Compute the points which yield the shortest distance given t.
- static void compute_t (VectorXd &t, const primitive_t &p_a, const primitive_t &p_b)

    Compute the t values based on the primitives (and their state).

9.1.1 Detailed Description

Public API.

This is the main public API which should be used.

9.1.2 Member Function Documentation

9.1.2.1 compute_D() [1/2]

```
static double DCA::API::compute_D (
            const primitive_t & p_a,
            const primitive_t & p_b ) [static]
```

Computes the shortest distance between two primitives.

Parameters

in	*p_a*	The first primitive
in	*p_b*	The second primitive

Returns

The shortest distance between both primitives.

9.1.2.2 compute_dDdS() [1/2]

```
static void DCA::API::compute_dDdS (
            Vector12d & dDdS,
            const primitive_t & p_a,
            const primitive_t & p_b ) [static]
```

Computes the first derivative of the shortest distance.

Computes the first derivative of the shortest distance between two primitives with respect to the state of the primitives.

Parameters

out	*dDdS*	The first derivative $\frac{dD}{dS}$
in	*p_a*	The first primitive
in	*p_b*	The second primitive

9.1.2.3 compute_d2DdS2() [1/2]

```
static void DCA::API::compute_d2DdS2 (
            Matrix12d & d2DdS2,
```

```
            const primitive_t & p_a,
            const primitive_t & p_b )  [static]
```

Computes the second derivative of the shortest distance.

Computes the second derivative of the shortest distance between two primitives with respect to the state of the primitives.

Parameters

out	*d2DdS2*	The second derivative $\frac{d^2D}{dS^2}$
in	*p_a*	The first primitive
in	*p_b*	The second primitive

9.1.2.4 compute_closest_points() [1/2]

```
static std::pair<Vector3d, Vector3d> DCA::API::compute_closest_points (
            const primitive_t & p_a,
            const primitive_t & p_b )  [static]
```

Compute the points which yield the shortest distance.

Returns a pair of points, where the distance between those points is the shortest between the primitives. The first returned point belongs to primitive p_a, the second belongs to primitive p_b.

Parameters

in	*p_a*	The first primitive
in	*p_b*	The second primitive

Returns

A pair of points, where the distance between those points is the shortest between the two primitives.

9.1.2.5 compute_D() [2/2]

```
static double DCA::API::compute_D (
            const primitive_t & p_a,
            const primitive_t & p_b,
            const VectorXd & t )  [static]
```

Computes the shortest distance between two primitives given t.

Parameters

in	p←_a	The first primitive
in	p←_b	The second primitive
in	t	The t values for the parameterization.

Returns

The shortest distance between both primitives.

9.1.2.6 compute_dDdS() [2/2]

```
static void DCA::API::compute_dDdS (
            Vector12d & dDdS,
            const primitive_t & p_a,
            const primitive_t & p_b,
            const VectorXd & t ) [static]
```

Computes the first derivative of the shortest distance given t.

Computes the first derivative of the shortest distance between two primitives with respect to the state of the primitives.

Parameters

out	dDdS	The first derivative $\frac{dD}{dS}$
in	p_a	The first primitive
in	p_b	The second primitive
in	t	The t values for the parameterization.

9.1.2.7 compute_d2DdS2() [2/2]

```
static void DCA::API::compute_d2DdS2 (
            Matrix12d & d2DdS2,
            const primitive_t & p_a,
            const primitive_t & p_b,
            const VectorXd & t ) [static]
```

Computes the second derivative of the shortest distance given t.

Computes the second derivative of the shortest distance between two primitives with respect to the state of the primitives.

Parameters

out	*d2DdS2*	The second derivative $\frac{d^2D}{dS^2}$
in	*p_a*	The first primitive
in	*p_b*	The second primitive
in	*t*	The t values for the parameterization.

9.1.2.8 compute_closest_points() [2/2]

```
static std::pair<Vector3d, Vector3d> DCA::API::compute_closest_points (
            const primitive_t & p_a,
            const primitive_t & p_b,
            const VectorXd & t )  [static]
```

Compute the points which yield the shortest distance given t.

Returns a pair of points, where the distance between those points is the shortest between the primitives. The first returned point belongs to primitive p_a, the second belongs to primitive p_b.

Parameters

in	*p_a*	The first primitive
in	*p_b*	The second primitive
in	*t*	The t values for the parameterization.

Returns

A pair of points, where the distance between those points is the shortest between the two primitives.

9.1.2.9 compute_t()

```
static void DCA::API::compute_t (
            VectorXd & t,
            const primitive_t & p_a,
            const primitive_t & p_b )  [static]
```

Compute the t values based on the primitives (and their state).

Computes the t values based on the state of the primitives. The values represent the points which will yield the shortest distance between the two primitives.

Parameters

out	*t*	The t values which represent the parameterization of the shortest distance.
in	*p_a*	The first primitive
in	*p_b*	The second primitive

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

- API.h

9.2 DCA::Box Class Reference

Definition of a Box.

```
#include <Primitives.h>
```

Public Member Functions

- Box (const Vector3d &center, const Matrix3d &orientation, const Vector3d &dimensions, const double &safetyMargin=0.001)
- ~Box ()=default

  Default deconstructor.
- Vector3d compute_P (const Vector6d &s, const VectorXd &t) const override

  Compute a point on the primitive.
- Eigen::Matrix< double, 3, 6 > compute_dPdS (const Vector6d &s, const VectorXd &t) const override

  Compute the derivative of a point on the primitive with respect to s.
- Eigen::Matrix< double, 3, -1 > compute_dPdT (const Vector6d &s, const VectorXd &t) const override

  Compute the derivative of a point on the primitive with respect to t.
- std::array< Eigen::Matrix< double, 3, 6 >, 6 > compute_d2PdS2 (const Vector6d &s, const VectorXd &t) const override

  Compute the second derivative of a point on the primitive with respect to s.
- std::vector< Eigen::Matrix< double, 3, 6 > > compute_d2PdSdT (const Vector6d &s, const VectorXd &t) const override

  Compute the mixed second derivative of a point on the primitive with respect to s and t.
- Vector6d get_s () const override

  Helper to get the state.
- int SIZE_T () const override

  Helper to get the size of t.
- double get_largest_dimension_from_center () const override

  Get the largest dimension.

Public Attributes

- Vector3d center

  The center of mass of the box.
- Matrix3d orientation

  The orientation of the box.
- Vector3d dimensions

  Convention: local dimensions in x, y, z direction.

Private Member Functions

- std::tuple< Vector3d, Vector3d, Vector3d > getLocalVectors () const

  Get vectors from the coordinates.

Additional Inherited Members

9.2.1 Detailed Description

Definition of a Box.

The state of a box is as follows: (x1, y1, z1, th1, th2, th3), which means that it is center of mass and the exponential map theta.

The parameterization of a box is as follows: (t1, t2, t3), which means, the parameterisation of a box.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 Box()

```
DCA::Box::Box (
            const Vector3d & center,
            const Matrix3d & orientation,
            const Vector3d & dimensions,
            const double & safetyMargin = 0.001 )
```

Create a box primitive.

Parameters

in	*center*	The center of mass.
in	*orientation*	The orientation of the box.
in	*dimensions*	The three dimensions of the box in x, y and z direction.
in	*safetyMargin*	The safety margin of the box.

Exceptions

std::runtime_error	If safety margin < 0.
std::runtime_error	If any dimension < 0.
std::runtime_error	If the orientation is invalid.

9.2.2.2 ∼Box()

```
DCA::Box::∼Box ( ) [default]
```

Default deconstructor.

9.2.3 Member Function Documentation

9.2.3.1 compute_P()

```
Vector3d DCA::Box::compute_P (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute a point on the primitive.

Compute a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The point.

Implements DCA::Primitive.

9.2.3.2 compute_dPdS()

```
Eigen::Matrix<double, 3, 6> DCA::Box::compute_dPdS (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the derivative of a point on the primitive with respect to s.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{ds}$.

Implements DCA::Primitive.

9.2.3.3 compute_dPdT()

```
Eigen::Matrix<double, 3, -1> DCA::Box::compute_dPdT (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the derivative of a point on the primitive with respect to t.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{dt}$.

In the case of a box, this has size 3x3.

Implements DCA::Primitive.

9.2.3.4 compute_d2PdS2()

```
std::array<Eigen::Matrix<double, 3, 6>, 6> DCA::Box::compute_d2PdS2 (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the second derivative of a point on the primitive with respect to s.

Compute the second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The second derivative $\frac{d^2P}{ds^2}$ as an array (size 6).

Implements DCA::Primitive.

9.2.3.5 compute_d2PdSdT()

```
std::vector<Eigen::Matrix<double, 3, 6> > DCA::Box::compute_d2PdSdT (
            const Vector6d & s,
            const VectorXd & t ) const [override], [virtual]
```

Compute the mixed second derivative of a point on the primitive with respect to s and t.

Compute the mixed second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{d^2P}{dsdt}$ as a vector (size t).

In the case of a box, this returns a vector with size 3.

Implements DCA::Primitive.

9.2.3.6 get_s()

```
Vector6d DCA::Box::get_s ( ) const [override], [virtual]
```

Helper to get the state.

Returns

The state of the primitive.

Implements DCA::Primitive.

9.2.3.7 SIZE_T()

```
int DCA::Box::SIZE_T ( ) const [override], [virtual]
```

Helper to get the size of t.

Returns

The number of dimensions needed for parameterization.

In the case of a rectangle, this returns 3.

Implements DCA::Primitive.

9.2.3.8 get_largest_dimension_from_center()

```
double DCA::Box::get_largest_dimension_from_center ( ) const  [override], [virtual]
```

Get the largest dimension.

Helper for pair generation.

Returns

The largest dimension from the center point.

Implements DCA::Primitive.

9.2.3.9 getLocalVectors()

```
std::tuple<Vector3d, Vector3d, Vector3d> DCA::Box::getLocalVectors ( ) const  [private]
```

Get vectors from the coordinates.

Returns

The local vectors.

9.2.4 Member Data Documentation

9.2.4.1 center

```
Vector3d DCA::Box::center
```

The center of mass of the box.

9.2.4.2 orientation

```
Matrix3d DCA::Box::orientation
```

The orientation of the box.

9.2.4.3 dimensions

```
Vector3d DCA::Box::dimensions
```

Convention: local dimensions in x, y, z direction.

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

- Primitives.h

9.3 DCA::Capsule Class Reference

Definition of a Capsule.

```
#include <Primitives.h>
```

Public Member Functions

- Capsule (const Vector3d &startPosition, const Vector3d &endPosition, const double &radius)
- ∼Capsule ()=default

  Default deconstructor.
- Vector3d compute_P (const Vector6d &s, const VectorXd &t) const override

  Compute a point on the primitive.
- Eigen::Matrix< double, 3, 6 > compute_dPdS (const Vector6d &s, const VectorXd &t) const override

  Compute the derivative of a point on the primitive with respect to s.
- Eigen::Matrix< double, 3, -1 > compute_dPdT (const Vector6d &s, const VectorXd &t) const override

  Compute the derivative of a point on the primitive with respect to t.
- std::array< Eigen::Matrix< double, 3, 6 >, 6 > compute_d2PdS2 (const Vector6d &s, const VectorXd &t) const override

  Compute the second derivative of a point on the primitive with respect to s.
- std::vector< Eigen::Matrix< double, 3, 6 > > compute_d2PdSdT (const Vector6d &s, const VectorXd &t) const override

  Compute the mixed second derivative of a point on the primitive with respect to s and t.
- Vector6d get_s () const override

  Helper to get the state.
- int SIZE_T () const override

  Helper to get the size of t.
- double get_largest_dimension_from_center () const override

  Get the largest dimension.

Public Attributes

- Vector3d startPosition

  The start position of the capsule in 3d space.
- Vector3d endPosition

  The end position of the capsule in 3d space.

Additional Inherited Members

9.3.1 Detailed Description

Definition of a Capsule.

The state of a capsule is as follows: (x1, y1, z1, x2, y2, z2), which means that it is the start and end point of the underlying line segment, stacked.

The parameterization of a capsule is as follows: (t1), which means, the position on the underlying line segment from -1 to 1.

9.3.2 Constructor & Destructor Documentation

9.3.2.1 Capsule()

```
DCA::Capsule::Capsule (
            const Vector3d & startPosition,
            const Vector3d & endPosition,
            const double & radius )
```

Create a capsule primitive.

Parameters

in	*startPosition*	The start position of the underlying line segment.
in	*endPosition*	The end position of the underlying line segment.
in	*radius*	The safety margin around the center.

Exceptions

std::runtime_error	If the radius is smaller than the distance between start and end.

9.3.2.2 ∼Capsule()

```
DCA::Capsule::∼Capsule ( ) [default]
```

Default deconstructor.

9.3.3 Member Function Documentation

9.3.3.1 compute_P()

```
Vector3d DCA::Capsule::compute_P (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute a point on the primitive.

Compute a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The point.

Implements DCA::Primitive.

9.3.3.2 compute_dPdS()

```
Eigen::Matrix<double, 3, 6> DCA::Capsule::compute_dPdS (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the derivative of a point on the primitive with respect to s.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{ds}$.

Implements DCA::Primitive.

9.3.3.3 compute_dPdT()

```
Eigen::Matrix<double, 3, -1> DCA::Capsule::compute_dPdT (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the derivative of a point on the primitive with respect to t.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{dt}$.

In the case of a capsule, this has size 3x1.

Implements DCA::Primitive.

9.3.3.4 compute_d2PdS2()

```
std::array<Eigen::Matrix<double, 3, 6>, 6> DCA::Capsule::compute_d2PdS2 (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the second derivative of a point on the primitive with respect to s.

Compute the second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The second derivative $\frac{d^2P}{ds^2}$ as an array (size 6).

Implements DCA::Primitive.

9.3.3.5 compute_d2PdSdT()

```
std::vector<Eigen::Matrix<double, 3, 6> > DCA::Capsule::compute_d2PdSdT (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the mixed second derivative of a point on the primitive with respect to s and t.

Compute the mixed second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{d^2P}{dsdt}$ as a vector (size t).

In the case of a capsule, this returns a vector with one matrix.

Implements DCA::Primitive.

9.3.3.6 get_s()

```
Vector6d DCA::Capsule::get_s ( ) const  [override], [virtual]
```

Helper to get the state.

Returns

The state of the primitive.

Implements DCA::Primitive.

9.3.3.7 SIZE_T()

```
int DCA::Capsule::SIZE_T ( ) const  [override], [virtual]
```

Helper to get the size of t.

Returns

The number of dimensions needed for parameterization.

In the case of a capsule, this returns 1.

Implements DCA::Primitive.

9.3.3.8 get_largest_dimension_from_center()

```
double DCA::Capsule::get_largest_dimension_from_center ( ) const  [override], [virtual]
```

Get the largest dimension.

Helper for pair generation.

Returns

The largest dimension from the center point.

Implements DCA::Primitive.

9.3.4 Member Data Documentation

9.3.4.1 startPosition

```
Vector3d DCA::Capsule::startPosition
```

The start position of the capsule in 3d space.

9.3.4.2 endPosition

```
Vector3d DCA::Capsule::endPosition
```

The end position of the capsule in 3d space.

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

- Primitives.h

9.4 DCA::ExpCoords Class Reference

Represents Rotations using exponential coordinates.

```
#include <ExpCoords.h>
```

Static Public Member Functions

- static Vector3d get_theta (const Matrix3d &R)

    Get θ from a rotation matrix.
- static Matrix3d get_dThetadRi (const Matrix3d &R, const int &index)

    Get $\frac{d\theta}{dR_i}$.
- static Matrix3d get_R (const Vector3d &theta)

    Get a rotation matrix from given exponential map.
- static Matrix3d get_dRi (const Vector3d &theta, int i)

    Returns the Jacobian of R.
- static Matrix3d get_ddR_i_j (const Vector3d &theta, int i, int j)

    Returns the derivative of the Jacobian of R at an index.
- static Vector3d get_w (const Vector3d &theta, const Vector3d &x)

    Returns world coordinates for a vector.
- static Matrix3d get_dwdr (const Vector3d &theta, const Vector3d &x)

    Returns the first derivative of ExpCoords::get_w.
- static Matrix3d get_ddwdr_dr1 (const Vector3d &theta, const Vector3d &x)

    Returns $\frac{d^2w}{dRdR_1}$.
- static Matrix3d get_ddwdr_dr2 (const Vector3d &theta, const Vector3d &x)

    Returns $\frac{d^2w}{dRdR_2}$.
- static Matrix3d get_ddwdr_dr3 (const Vector3d &theta, const Vector3d &x)

    Returns $\frac{d^2w}{dRdR_3}$.

Static Private Member Functions

- static Matrix3d getSkewSymmetricMatrix (const Vector3d &v)

    Get the skew symmetric matrix for a given vector.
- static double safeACOS (double x)

    Safe arcus cosinus.
- static double safeDACOS (double x)

    Safe derivative of arcus cosinus.

Static Private Attributes

- constexpr static const double tol = 1e-8

    Tolerance for avoiding singularities of Rodrigues' formula.
- constexpr static const double _PI = 3.14159265359

    Helper.

9.4.1 Detailed Description

Represents Rotations using exponential coordinates.

See e.g. `https://arxiv.org/pdf/1312.0788.pdf`

9.4.2 Member Function Documentation

9.4.2.1 get_theta()

```
static Vector3d DCA::ExpCoords::get_theta (
            const Matrix3d & R )  [inline], [static]
```

Get θ from a rotation matrix.

Returns the exponential map parameterization

Parameters

in	*R*	The rotation matrix

Returns

The exponential map parameterization.

9.4.2.2 get_dThetadRi()

```
static Matrix3d DCA::ExpCoords::get_dThetadRi (
            const Matrix3d & R,
            const int & index )  [inline], [static]
```

Get $\frac{d\theta}{dR_i}$.

Returns the derivative of exponential coorinates with respect to a column of R

Parameters

in	*R*	The rotation matrix
in	*index*	The column to use. Must be either 0, 1 or 2.

Returns

The exponential map parameterization.

Exceptions

std::runtime_error	If the index is out of bounds.

9.4.2.3 get_R()

```
static Matrix3d DCA::ExpCoords::get_R (
            const Vector3d & theta )  [inline], [static]
```

Get a rotation matrix from given exponential map.

Parameters

in	*theta*	θ

Returns

The rotation matrix which is represented by θ.

9.4.2.4 get_dRi()

```
static Matrix3d DCA::ExpCoords::get_dRi (
            const Vector3d & theta,
            int i ) [inline], [static]
```

Returns the Jacobian of R.

Returns a Jacobian that tells us how the rotation matrix changes with respect to the exponential map parameters that define it.

Parameters

in	*theta*	θ
in	*i*	The index (column).

Returns

$\frac{dR}{d\theta_i}$

9.4.2.5 get_ddR_i_j()

```
static Matrix3d DCA::ExpCoords::get_ddR_i_j (
            const Vector3d & theta,
            int i,
            int j ) [inline], [static]
```

Returns the derivative of the Jacobian of R at an index.

Returns the derivative of the Jacobian $\frac{dR}{d\theta_i}$ with respect to the exponential map parameters at index j.

Parameters

in	*theta*	θ
in	*i*	The first index.
in	*j*	The second index.

Returns

$\frac{d^2 R}{d\theta_i d\theta_j}$

Exceptions

std::runtime_error	(only release) if either i or j is not in [0,2].

9.4.2.6 get_w()

```
static Vector3d DCA::ExpCoords::get_w (
            const Vector3d & theta,
            const Vector3d & x ) [inline], [static]
```

Returns world coordinates for a vector.

Returns the world coordinates for the vector x that is expressed in local coordinates.

Parameters

in	*theta*	The exponential map representation.
in	*x*	The vector in local coordinates.

Returns

The world coordinates for x.

9.4.2.7 get_dwdr()

```
static Matrix3d DCA::ExpCoords::get_dwdr (
            const Vector3d & theta,
            const Vector3d & x ) [inline], [static]
```

Returns the first derivative of ExpCoords::get_w.

Returns a matrix that tells us how w changes with respect to the orientation.

Parameters

in	*theta*	The exponential map representation.
in	*x*	The vector in local coordinates.

Returns

The derivative of w with respect to the orientation.

9.4.2.8 get_ddwdr_dr1()

```
static Matrix3d DCA::ExpCoords::get_ddwdr_dr1 (
            const Vector3d & theta,
            const Vector3d & x )  [inline], [static]
```

Returns $\frac{d^2w}{dRdR_1}$.

Returns the second derivative of ExpCoords::get_w with respect to the first component of the orientation.

Parameters

in	*theta*	The exponential map representation.
in	*x*	The local coordinates of the point.

Returns

The second derivative.

9.4.2.9 get_ddwdr_dr2()

```
static Matrix3d DCA::ExpCoords::get_ddwdr_dr2 (
            const Vector3d & theta,
            const Vector3d & x )  [inline], [static]
```

Returns $\frac{d^2w}{dRdR_2}$.

Returns the second derivative of ExpCoords::get_w with respect to the second component of the orientation.

Parameters

in	*theta*	The exponential map representation.
in	*x*	The local coordinates of the point.

Returns

The second derivative.

9.4.2.10 get_ddwdr_dr3()

```
static Matrix3d DCA::ExpCoords::get_ddwdr_dr3 (
            const Vector3d & theta,
            const Vector3d & x )  [inline], [static]
```

Returns $\frac{d^2w}{dRdR_3}$.

Returns the second derivative of ExpCoords::get_w with respect to the third component of the orientation.

Parameters

in	*theta*	The exponential map representation.
in	*x*	The local coordinates of the point.

Returns

The second derivative.

9.4.2.11 getSkewSymmetricMatrix()

```
static Matrix3d DCA::ExpCoords::getSkewSymmetricMatrix (
            const Vector3d & v )  [inline], [static], [private]
```

Get the skew symmetric matrix for a given vector.

Parameters

in	*v*	The vector.

Returns

The skew symmetric matrix.

9.4.2.12 safeACOS()

```
static double DCA::ExpCoords::safeACOS (
            double x )  [inline], [static], [private]
```

Safe arcus cosinus.

Parameters

in	*x*	The value.

Returns

safe arcus cosinus.

9.4.2.13 safeDACOS()

```
static double DCA::ExpCoords::safeDACOS (
            double x )  [inline], [static], [private]
```

Safe derivative of arcus cosinus.

Parameters

in	*x*	The value.

Returns

Safe derivative of the arcus cosinus.

9.4.3 Member Data Documentation

9.4.3.1 tol

```
constexpr static const double DCA::ExpCoords::tol = 1e-8  [static], [constexpr], [private]
```

Tolerance for avoiding singularities of Rodrigues' formula.

9.4.3.2 _PI

```
constexpr static const double DCA::ExpCoords::_PI = 3.14159265359  [static], [constexpr],
[private]
```

Helper.

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

- ExpCoords.h

9.5 DCA::FiniteDifference Class Reference

Helper for finite differences (to test derivatives).

```
#include <FiniteDifference.h>
```

9.5.1 Detailed Description

Helper for finite differences (to test derivatives).

Attention

Not documented.

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

- FiniteDifference.h

9.6 DCA::Logger Class Reference

Logger class.

`#include <Logger.h>`

Public Types

- enum PRINT_COLOR {
  RED , GREEN , YELLOW , BLUE ,
  MAGENTA , CYAN , DEFAULT }

  Possible color values.

Static Public Member Functions

- static void print (PRINT_COLOR color, const char *fmt,...)

  Print a message.

9.6.1 Detailed Description

Logger class.

Writes to stout.

9.6.2 Member Enumeration Documentation

9.6.2.1 PRINT_COLOR

`enum DCA::Logger::PRINT_COLOR`

Possible color values.

Enumerator

RED	
GREEN	
YELLOW	
BLUE	
MAGENTA	
CYAN	
DEFAULT	

9.6.3 Member Function Documentation

9.6.3.1 print()

```
static void DCA::Logger::print (
            PRINT_COLOR color,
            const char * fmt,
             ... ) [inline], [static]
```

Print a message.

Parameters

in	*color*	The color to use.
in	*fmt*	The thing to print, potentially containing format strings.

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

- Logger.h

9.7 DCA::NeighborsPairGenerator Class Reference

This generator computes the pairs which are in a certain threshold from each other. It does so by computing a single position for each primitive and selecting pairs based on the distance.

`#include <Pair.h>`

Public Member Functions

- NeighborsPairGenerator (const double &radius)

  Construct this generator with a given radius.
- std::vector< pair_t > generate (const std::vector< primitive_t > &primitives) const

  This function generates the pairs given the primitives, where the pairs are in a certain distance from each other.
- std::vector< pair_t > generate (const std::vector< primitive_t > &primitives_a, const std::vector< primitive_t > &primitives_b) const

  This function generates pairs given two vectors of primitives, where the pairs are in a certain distance from each other and are in separate vectors.

Private Member Functions

- double estimate_distance (const primitive_t &p_A, const primitive_t &p_B) const
  Helper function for generate.

Private Attributes

- double m_radius
  The radius.

9.7.1 Detailed Description

This generator computes the pairs which are in a certain threshold from each other. It does so by computing a single position for each primitive and selecting pairs based on the distance.

9.7.2 Constructor & Destructor Documentation

9.7.2.1 NeighborsPairGenerator()

```
DCA::NeighborsPairGenerator::NeighborsPairGenerator (
              const double & radius )
```

Construct this generator with a given radius.

Parameters

in	*radius*	The radius to search other primitives in.

9.7.3 Member Function Documentation

9.7.3.1 generate() [1/2]

```
std::vector<pair_t> DCA::NeighborsPairGenerator::generate (
              const std::vector< primitive_t > & primitives ) const
```

This function generates the pairs given the primitives, where the pairs are in a certain distance from each other.

The returned vector consists of pairs, where each pair holds two numbers: The indices of the corresponding primitives which were given.

Parameters

in	*primitives*	All primitives to generate the pairs from.

Returns

A vector of pairs of indices, where each index corresponds to a primitive in the primitives vector.

9.7.3.2 generate() [2/2]

```
std::vector<pair_t> DCA::NeighborsPairGenerator::generate (
            const std::vector< primitive_t > & primitives_a,
            const std::vector< primitive_t > & primitives_b ) const
```

This function generates pairs given two vectors of primitives, where the pairs are in a certain distance from each other and are in separate vectors.

The returned vector consists of pairs, where each pair holds two numbers: The indices of the corresponding primitives which were given. The first index corresponds to the primitives_a vector, the second to the primitives_b vector.

Parameters

in	*primitives_a*	The first set of primitives.
in	*primitives_b*	The second set of primitives.

Returns

A vector of pairs of indices, where the first index corresponds to a primitive in the primitives_a vector and the second index to the primitives_b vector.

Attention

Not all pairs are reported, only those between primitives_a and primitives_b!

9.7.3.3 estimate_distance()

```
double DCA::NeighborsPairGenerator::estimate_distance (
            const primitive_t & p_A,
            const primitive_t & p_B ) const [private]
```

Helper function for generate.

9.7.4 Member Data Documentation

9.7.4.1 m_radius

```
double DCA::NeighborsPairGenerator::m_radius [private]
```

The radius.

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

- Pair.h

9.8 DCA::NewtonOptimizer Class Reference

Simply newton optimizer class.

```
#include <NewtonOptimizer.h>
```

Public Member Functions

- NewtonOptimizer (const std::string &name="NewtonOptimizer", double solverResidual=1e-5, int maxLineSearchIterations=15)

    Construct a newton optimizer.
- ∼NewtonOptimizer ()=default

    Default deconstructor.
- bool optimize (VectorXd &t, const VectorXd &s, const Objective &objective, int maxIterations)

    Perform the optimization.

Public Attributes

- std::string name

    Name of the solver (mainly used for printing)
- double solverResidual

    Accurancy when solver states that it has converged (compared to the norm of the gradient)
- int maxLineSearchIterations

    Max number of line search iterations before giving up.
- double lineSearchStartValue = 1.0

    Start value of line search. If unsuccessful, this value is cut in half until a suitable solution is found.
- bool useDynamicRegularization = true

    If activated, we regularize the hessian in case we don't have a decending search direction.
- bool printInfo = false

    Print infos to console to observe solver progress.
- bool checkLinearSystemSolve = false

    Check if linear system has been solved successfully (i.e. the correct linear solver has been used)
- bool checkHessianRank = false

    Check if hessian is invertible.

Private Attributes

- double objectiveValue

  The objective value.
- VectorXd t_tmp

  Temporary t variable.
- VectorXd searchDir

  Current search direction.
- VectorXd gradient

  Current gradient.
- MatrixXd hessian

  Current hessian.

9.8.1 Detailed Description

Simply newton optimizer class.

9.8.2 Constructor & Destructor Documentation

9.8.2.1 NewtonOptimizer()

```
DCA::NewtonOptimizer::NewtonOptimizer (
            const std::string & name = "NewtonOptimizer",
            double solverResidual = 1e-5,
            int maxLineSearchIterations = 15 )  [inline]
```

Construct a newton optimizer.

Parameters

in	*name*	The name of the optimizer.
in	*solverResidual*	The solver residual.
in	*maxLineSearchIterations*	The maximum number of line search iterations.

9.8.2.2 ∼NewtonOptimizer()

```
DCA::NewtonOptimizer::~NewtonOptimizer ( )  [default]
```

Default deconstructor.

9.8.3 Member Function Documentation

9.8.3.1 optimize()

```
bool DCA::NewtonOptimizer::optimize (
            VectorXd & t,
            const VectorXd & s,
            const Objective & objective,
            int maxIterations )
```

Perform the optimization.

Parameters

out	*t*	The resulting t values.
in	*s*	The state of two primitives.
in	*objective*	The objective to optimize.
in	*maxIterations*	The maximum number of iterations.

Returns

True, if the solver has converged.

9.8.4 Member Data Documentation

9.8.4.1 name

```
std::string DCA::NewtonOptimizer::name
```

Name of the solver (mainly used for printing)

9.8.4.2 solverResidual

```
double DCA::NewtonOptimizer::solverResidual
```

Accurancy when solver states that it has converged (compared to the norm of the gradient)

9.8.4.3 maxLineSearchIterations

```
int DCA::NewtonOptimizer::maxLineSearchIterations
```

Max number of line search iterations before giving up.

9.8.4.4 lineSearchStartValue

```
double DCA::NewtonOptimizer::lineSearchStartValue = 1.0
```

Start value of line search. If unsuccessful, this value is cut in half until a suitable solution is found.

9.8.4.5 useDynamicRegularization

```
bool DCA::NewtonOptimizer::useDynamicRegularization = true
```

If activated, we regularize the hessian in case we don't have a decending search direction.

9.8.4.6 printInfo

```
bool DCA::NewtonOptimizer::printInfo = false
```

Print infos to console to observe solver progress.

9.8.4.7 checkLinearSystemSolve

```
bool DCA::NewtonOptimizer::checkLinearSystemSolve = false
```

Check if linear system has been solved successfully (i.e. the correct linear solver has been used)

9.8.4.8 checkHessianRank

```
bool DCA::NewtonOptimizer::checkHessianRank = false
```

Check if hessian is invertible.

9.8.4.9 objectiveValue

```
double DCA::NewtonOptimizer::objectiveValue  [private]
```

The objective value.

9.8.4.10 t_tmp

```
VectorXd DCA::NewtonOptimizer::t_tmp  [private]
```

Temporary t variable.

9.8.4.11 searchDir

```
VectorXd DCA::NewtonOptimizer::searchDir  [private]
```

Current search direction.

9.8.4.12 gradient

```
VectorXd DCA::NewtonOptimizer::gradient  [private]
```

Current gradient.

9.8.4.13 hessian

```
MatrixXd DCA::NewtonOptimizer::hessian  [private]
```

Current hessian.

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

- NewtonOptimizer.h

9.9 DCA::Objective Class Reference

The main objective which used for computing distances and their derivatives.

```
#include <Objective.h>
```

Public Member Functions

- Objective (primitive_t primitive_A, primitive_t primitive_B)

  Create an objective for a pair of primitives.
- ∼Objective ()=default

  Default deconstructor.
- std::pair< Vector3d, Vector3d > compute_Ps (const VectorXd &s, const VectorXd &t) const

  Compute points on the primitives based on s and t.
- std::pair< int, int > get_sizes_t () const

  Return both sizes of t.
- int get_total_size_t () const

  Return the sum of t sizes.

D

D and its derivatives

- double compute_D (const VectorXd &s, const VectorXd &t) const

  Compute the distance.
- void compute_pDpT (VectorXd &pDpT, const VectorXd &s, const VectorXd &t) const

  Compute the partial derivative of the distance w.r.t. t.
- void compute_pDpS (VectorXd &pDpS, const VectorXd &s, const VectorXd &t) const

  Compute the partial derivative of the distance w.r.t. s.
- void compute_p2DpT2 (MatrixXd &p2DpT2, const VectorXd &s, const VectorXd &t) const

  Compute the second partial derivative of the distance w.r.t. t.
- void compute_p2DpS2 (MatrixXd &p2DpS2, const VectorXd &s, const VectorXd &t) const

  Compute the second partial derivative of the distance w.r.t. s.
- void compute_p2DpTpS (MatrixXd &p2DpTpS, const VectorXd &s, const VectorXd &t) const

  Compute the mixed partial derivative of the distance w.r.t. t and s.

O

O and its derivatives

- double compute_O (const VectorXd &s, const VectorXd &t) const

  Compute the objective value.
- void compute_pOpT (VectorXd &pOpT, const VectorXd &s, const VectorXd &t) const

  Compute the partial derivative of the objective value w.r.t. t.
- void compute_pOpS (VectorXd &pOpS, const VectorXd &s, const VectorXd &t) const

  Compute the partial derivative of the objective value w.r.t. s.
- void compute_p2OpT2 (MatrixXd &p2OpT2, const VectorXd &s, const VectorXd &t) const

  Compute the second partial derivative of the objective value w.r.t. t.
- void compute_p2OpS2 (MatrixXd &p2OpS2, const VectorXd &s, const VectorXd &t) const

  Compute the second partial derivative of the objective value w.r.t. s.
- void compute_p2OpTpS (MatrixXd &p2OpTpS, const VectorXd &s, const VectorXd &t) const

  Compute the mixed partial derivative of the objective value w.r.t. t and s.

Static Public Member Functions

- static std::string get_primitives_description (primitive_t primitive_A, primitive_t primitive_B)

  Get a string of both primitive descriptions.

Public Attributes

- double regularizerWeight = 0.01

  Regularitaion wheight for the objective.
- double constraintWeight = 10.0

  Constraint weight for the objective.
- SoftUnilateralConstraint unilateralConstraint = SoftUnilateralConstraint(0.0, 1.0, 1e-3)

  The soft unilateral constraint.
- const primitive_t primitive_A

  The first primitive.
- const primitive_t primitive_B

  The seonc primitive.

Private Types

- enum PRIMITIVE { A , B }

  Helper for the first or second primitive.

Private Member Functions

- Vector6d get_s (const VectorXd &s, PRIMITIVE P) const

  Get s for a primitive.
- VectorXd get_t (const VectorXd &t, PRIMITIVE P) const

  Get s for a primitive.
- void check_inputs (const VectorXd &s, const VectorXd &t) const

  Check s and t.

9.9.1 Detailed Description

The main objective which used for computing distances and their derivatives.

9.9.2 Member Enumeration Documentation

9.9.2.1 PRIMITIVE

```
enum DCA::Objective::PRIMITIVE  [private]
```

Helper for the first or second primitive.

Enumerator

A	
B	

9.9.3 Constructor & Destructor Documentation

9.9.3.1 Objective()

```
DCA::Objective::Objective (
            primitive_t primitive_A,
            primitive_t primitive_B ) [inline]
```

Create an objective for a pair of primitives.

Parameters

in	*primitive_A*	The first primitive.
in	*primitive_B*	The second primitive.

9.9.3.2 ∼Objective()

```
DCA::Objective::~Objective ( ) [default]
```

Default deconstructor.

9.9.4 Member Function Documentation

9.9.4.1 compute_Ps()

```
std::pair<Vector3d, Vector3d> DCA::Objective::compute_Ps (
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute points on the primitives based on s and t.

Parameters

in	*s*	The state of both primitives, stacked.
in	*t*	The parameterization of both primitives, stacked.

9.9.4.2 compute_D()

```
double DCA::Objective::compute_D (
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the distance.

Parameters

in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

Returns

The distance between the two primitives.

9.9.4.3 compute_pDpT()

```
void DCA::Objective::compute_pDpT (
            VectorXd & pDpT,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the partial derivative of the distance w.r.t. t.

Parameters

out	*pDpT*	$\frac{\partial D}{\partial t}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.4 compute_pDpS()

```
void DCA::Objective::compute_pDpS (
            VectorXd & pDpS,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the partial derivative of the distance w.r.t. s.

Parameters

out	*pDpS*	$\frac{\partial D}{\partial s}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.5 compute_p2DpT2()

```
void DCA::Objective::compute_p2DpT2 (
            MatrixXd & p2DpT2,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the second partial derivative of the distance w.r.t. t.

Parameters

out	*p2DpT2*	$\frac{\partial^2 D}{\partial t^2}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.6 compute_p2DpS2()

```
void DCA::Objective::compute_p2DpS2 (
            MatrixXd & p2DpS2,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the second partial derivative of the distance w.r.t. s.

Parameters

out	*p2DpS2*	$\frac{\partial^2 D}{\partial s^2}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.7 compute_p2DpTpS()

```
void DCA::Objective::compute_p2DpTpS (
            MatrixXd & p2DpTpS,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the mixed partial derivative of the distance w.r.t. t and s.

Parameters

out	*p2DpTpS*	$\frac{\partial^2 D}{\partial t \partial s}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.8 compute_O()

```
double DCA::Objective::compute_O (
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the objective value.

Parameters

in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

Returns

The objective value.

9.9.4.9 compute_pOpT()

```
void DCA::Objective::compute_pOpT (
            VectorXd & pOpT,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the partial derivative of the objective value w.r.t. t.

Parameters

out	*pOpT*	$\frac{\partial O}{\partial t}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.10 compute_pOpS()

```
void DCA::Objective::compute_pOpS (
            VectorXd & pOpS,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the partial derivative of the objective value w.r.t. s.

Parameters

out	*pOpS*	$\frac{\partial O}{\partial s}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.11 compute_p2OpT2()

```
void DCA::Objective::compute_p2OpT2 (
            MatrixXd & p2OpT2,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the second partial derivative of the objective value w.r.t. t.

Parameters

out	*p2OpT2*	$\frac{\partial^2 O}{\partial t^2}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.12 compute_p2OpS2()

```
void DCA::Objective::compute_p2OpS2 (
            MatrixXd & p2OpS2,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the second partial derivative of the objective value w.r.t. s.

Parameters

out	*p2OpS2*	$\frac{\partial^2 O}{\partial s^2}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.13 compute_p2OpTpS()

```
void DCA::Objective::compute_p2OpTpS (
            MatrixXd & p2OpTpS,
```

```
const VectorXd & s,
const VectorXd & t ) const
```

Compute the mixed partial derivative of the objective value w.r.t. t and s.

Parameters

out	*p2OpTpS*	$\frac{\partial^2 O}{\partial t \partial s}$
in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives.

9.9.4.14 get_sizes_t()

```
std::pair<int, int> DCA::Objective::get_sizes_t ( ) const
```

Return both sizes of t.

Returns

Both sizes of t.

9.9.4.15 get_total_size_t()

```
int DCA::Objective::get_total_size_t ( ) const
```

Return the sum of t sizes.

Returns

The sum of t sizes.

9.9.4.16 get_primitives_description()

```
static std::string DCA::Objective::get_primitives_description (
            primitive_t primitive_A,
            primitive_t primitive_B ) [static]
```

Get a string of both primitive descriptions.

Parameters

in	*primitive↩_A*	The first primitive.
in	*primitive↩_B*	The second primitive.

Returns

The string.

9.9.4.17 get_s()

```
Vector6d DCA::Objective::get_s (
            const VectorXd & s,
            PRIMITIVE P ) const  [private]
```

Get s for a primitive.

Parameters

in	*s*	The stacked s vector.
in	*P*	the primitive to use.

Returns

The state belonging to the given primitive.

9.9.4.18 get_t()

```
VectorXd DCA::Objective::get_t (
            const VectorXd & t,
            PRIMITIVE P ) const  [private]
```

Get s for a primitive.

Parameters

in	*t*	The stacked s vector.
in	*P*	the primitive to use.

Returns

The parameterization belonging to the given primitive.

9.9.4.19 check_inputs()

```
void DCA::Objective::check_inputs (
            const VectorXd & s,
            const VectorXd & t ) const  [private]
```

Check s and t.

Parameters

in	*s*	The stacked s vector.
in	*t*	the stacked t vector.

Exceptions

std::runtime_error	If the size is wrong.

9.9.5 Member Data Documentation

9.9.5.1 regularizerWeight

```
double DCA::Objective::regularizerWeight = 0.01
```

Regularitaion wheight for the objective.

9.9.5.2 constraintWeight

```
double DCA::Objective::constraintWeight = 10.0
```

Constraint weight for the objective.

9.9.5.3 unilateralConstraint

```
SoftUnilateralConstraint DCA::Objective::unilateralConstraint = SoftUnilateralConstraint(0.0,
1.0, 1e-3)
```

The soft unilateral constraint.

9.9.5.4 primitive_A

```
const primitive_t DCA::Objective::primitive_A
```

The first primitive.

9.9.5.5 primitive_B

```
const primitive_t DCA::Objective::primitive_B
```

The seonc primitive.

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

- Objective.h

9.10 DCA::PermutationPairGenerator Class Reference

This generator creates all possible permutations of pairs. This means, the amount of pairs created is $n^2/2$, where n is the number of primitives. Pairs are not returned twice (0, 1) and (1, 0).

```
#include <Pair.h>
```

Public Member Functions

- std::vector< pair_t > generate (const std::vector< primitive_t > &primitives) const
  This function generates all pairs given the primitives.

9.10.1 Detailed Description

This generator creates all possible permutations of pairs. This means, the amount of pairs created is $n^2/2$, where n is the number of primitives. Pairs are not returned twice (0, 1) and (1, 0).

9.10.2 Member Function Documentation

9.10.2.1 generate()

```
std::vector<pair_t> DCA::PermutationPairGenerator::generate (
            const std::vector< primitive_t > & primitives ) const
```

This function generates all pairs given the primitives.

The returned vector consists of pairs, where each pair holds two numbers: The indices of the corresponding primitives which were given.

Parameters

in	*primitives*	All primitives to generate the pairs from.

Returns

A vector of pairs of indices, where each index corresponds to a primitive in the primitives vector.

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

- Pair.h

9.11 DCA::Primitive Class Reference

Definition of a Primitive.

```
#include <Primitives.h>
```

Public Member Functions

- Primitive (const std::string &description, const double &safetyMargin)

  Create a primitive.
- virtual ∼Primitive ()=default

  Default deconstructor.
- virtual Vector3d compute_P (const Vector6d &s, const VectorXd &t) const =0

  Compute a point on the primitive.
- virtual Eigen::Matrix< double, 3, 6 > compute_dPdS (const Vector6d &s, const VectorXd &t) const =0

  Compute the derivative of a point on the primitive with respect to s.
- virtual Eigen::Matrix< double, 3, -1 > compute_dPdT (const Vector6d &s, const VectorXd &t) const =0

  Compute the derivative of a point on the primitive with respect to t.
- virtual std::array< Eigen::Matrix< double, 3, 6 >, 6 > compute_d2PdS2 (const Vector6d &s, const VectorXd &t) const =0

  Compute the second derivative of a point on the primitive with respect to s.
- virtual std::vector< Eigen::Matrix< double, 3, 6 > > compute_d2PdSdT (const Vector6d &s, const VectorXd &t) const =0

  Compute the mixed second derivative of a point on the primitive with respect to s and t.
- virtual Vector6d get_s () const =0

  Helper to get the state.
- virtual int SIZE_T () const =0

  Helper to get the size of t.
- Vector3d get_center_point () const

  Get the center point.
- virtual double get_largest_dimension_from_center () const =0

  Get the largest dimension.

Public Attributes

- double safetyMargin

  Internal storage of the safety margin.

Protected Member Functions

- void check_t (const VectorXd &t) const

    Check the size of t.

9.11.1 Detailed Description

Definition of a Primitive.

Each Primitive can be described by a 6-dimensional state and a n-dimensional parameterization vector.

This means, any convex primitive which can be described in a continuously differentiable manner can be implemented.

The inheritance on FiniteDifference is just to make the automatic derivative checking easier.

9.11.2 Constructor & Destructor Documentation

9.11.2.1 Primitive()

```
DCA::Primitive::Primitive (
            const std::string & description,
            const double & safetyMargin )
```

Create a primitive.

Parameters

in	*description*	The description of the primitive.
in	*safetyMargin*	The safety margin of the primitive.

Exceptions

std::runtime_error	if safety margin < 0

9.11.2.2 ∼Primitive()

```
virtual DCA::Primitive::~Primitive ( )  [virtual], [default]
```

Default deconstructor.

9.11.3 Member Function Documentation

9.11.3.1 compute_P()

```
virtual Vector3d DCA::Primitive::compute_P (
            const Vector6d & s,
            const VectorXd & t ) const  [pure virtual]
```

Compute a point on the primitive.

Compute a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The point.

Implemented in DCA::Box, DCA::Rectangle, DCA::Capsule, and DCA::Sphere.

9.11.3.2 compute_dPdS()

```
virtual Eigen::Matrix<double, 3, 6> DCA::Primitive::compute_dPdS (
            const Vector6d & s,
            const VectorXd & t ) const  [pure virtual]
```

Compute the derivative of a point on the primitive with respect to s.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{ds}$.

Implemented in DCA::Box, DCA::Rectangle, DCA::Capsule, and DCA::Sphere.

9.11.3.3 compute_dPdT()

```
virtual Eigen::Matrix<double, 3, -1> DCA::Primitive::compute_dPdT (
            const Vector6d & s,
            const VectorXd & t ) const  [pure virtual]
```

Compute the derivative of a point on the primitive with respect to t.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{dt}$.

Implemented in DCA::Box, DCA::Rectangle, DCA::Capsule, and DCA::Sphere.

9.11.3.4 compute_d2PdS2()

```
virtual std::array<Eigen::Matrix<double, 3, 6>, 6> DCA::Primitive::compute_d2PdS2 (
            const Vector6d & s,
            const VectorXd & t ) const  [pure virtual]
```

Compute the second derivative of a point on the primitive with respect to s.

Compute the second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The second derivative $\frac{d^2P}{ds^2}$ as an array (size 6).

Implemented in DCA::Box, DCA::Rectangle, DCA::Capsule, and DCA::Sphere.

9.11.3.5 compute_d2PdSdT()

```
virtual std::vector<Eigen::Matrix<double, 3, 6> > DCA::Primitive::compute_d2PdSdT (
            const Vector6d & s,
            const VectorXd & t ) const  [pure virtual]
```

Compute the mixed second derivative of a point on the primitive with respect to s and t.

Compute the mixed second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{d^2P}{dsdt}$ as a vector (size t).

Implemented in DCA::Box, DCA::Rectangle, DCA::Capsule, and DCA::Sphere.

9.11.3.6 get_s()

```
virtual Vector6d DCA::Primitive::get_s ( ) const [pure virtual]
```

Helper to get the state.

Returns

The state of the primitive.

Implemented in DCA::Box, DCA::Rectangle, DCA::Capsule, and DCA::Sphere.

9.11.3.7 SIZE_T()

```
virtual int DCA::Primitive::SIZE_T ( ) const [pure virtual]
```

Helper to get the size of t.

Returns

The number of dimensions needed for parameterization.

Implemented in DCA::Box, DCA::Rectangle, DCA::Capsule, and DCA::Sphere.

9.11.3.8 get_center_point()

```
Vector3d DCA::Primitive::get_center_point ( ) const
```

Get the center point.

Helper for pair generation.

Returns

The center point of the primitive.

9.11.3.9 get_largest_dimension_from_center()

```
virtual double DCA::Primitive::get_largest_dimension_from_center ( ) const  [pure virtual]
```

Get the largest dimension.

Helper for pair generation.

Returns

The largest dimension from the center point.

Implemented in DCA::Box, DCA::Rectangle, DCA::Capsule, and DCA::Sphere.

9.11.3.10 check_t()

```
void DCA::Primitive::check_t (
             const VectorXd & t ) const  [protected]
```

Check the size of t.

Parameters

in	*t*	The t vector.

Exceptions

std::runtime_error	If the size is wrong.

9.11.4 Member Data Documentation

9.11.4.1 safetyMargin

```
double DCA::Primitive::safetyMargin
```

Internal storage of the safety margin.

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

- Primitives.h

9.12 DCA::Rectangle Class Reference

Definition of a Rectangle.

```
#include <Primitives.h>
```

Public Member Functions

- Rectangle (const Vector3d &center, const Matrix3d &orientation, const Vector2d &dimensions, const double &safetyMargin=0.001)
- ∼Rectangle ()=default

  Default deconstructor.
- Vector3d compute_P (const Vector6d &s, const VectorXd &t) const override

  Compute a point on the primitive.
- Eigen::Matrix< double, 3, 6 > compute_dPdS (const Vector6d &s, const VectorXd &t) const override

  Compute the derivative of a point on the primitive with respect to s.
- Eigen::Matrix< double, 3, -1 > compute_dPdT (const Vector6d &s, const VectorXd &t) const override

  Compute the derivative of a point on the primitive with respect to t.
- std::array< Eigen::Matrix< double, 3, 6 >, 6 > compute_d2PdS2 (const Vector6d &s, const VectorXd &t) const override

  Compute the second derivative of a point on the primitive with respect to s.
- std::vector< Eigen::Matrix< double, 3, 6 > > compute_d2PdSdT (const Vector6d &s, const VectorXd &t) const override

  Compute the mixed second derivative of a point on the primitive with respect to s and t.
- Vector6d get_s () const override

  Helper to get the state.
- int SIZE_T () const override

  Helper to get the size of t.
- double get_largest_dimension_from_center () const override

  Get the largest dimension.

Public Attributes

- Vector3d center

  The center of mass of the rectangle.
- Matrix3d orientation

  The orientation of the rectangle.
- Vector2d dimensions

  Convention: local dimensions in x and z direction.

Private Member Functions

- std::pair< Vector3d, Vector3d > getLocalVectors () const

  Get vectors from the coordinates.

Additional Inherited Members

9.12.1 Detailed Description

Definition of a Rectangle.

The state of a rectangle is as follows: (x1, y1, z1, th1, th2, th3), which means that it is center of mass and the exponential map theta.

The parameterization of a rectangle is as follows: (t1, t2), which means, the parameterisation of a bounded plane.

9.12.2 Constructor & Destructor Documentation

9.12.2.1 Rectangle()

```
DCA::Rectangle::Rectangle (
            const Vector3d & center,
            const Matrix3d & orientation,
            const Vector2d & dimensions,
            const double & safetyMargin = 0.001 )
```

Create a rectangle primitive.

Parameters

in	*center*	The center of mass.
in	*orientation*	The orientation of the rectangle.
in	*dimensions*	The two dimensions of the rectangle in x and z direction.
in	*safetyMargin*	The safety margin of the rectangle.

Exceptions

std::runtime_error	If safety margin $<$ 0.
std::runtime_error	If any dimension $<$ 0.
std::runtime_error	If the orientation is invalid.

9.12.2.2 ∼Rectangle()

```
DCA::Rectangle::~Rectangle ( )  [default]
```

Default deconstructor.

9.12.3 Member Function Documentation

9.12.3.1 compute_P()

```
Vector3d DCA::Rectangle::compute_P (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute a point on the primitive.

Compute a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The point.

Implements DCA::Primitive.

9.12.3.2 compute_dPdS()

```
Eigen::Matrix<double, 3, 6> DCA::Rectangle::compute_dPdS (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the derivative of a point on the primitive with respect to s.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{ds}$.

Implements DCA::Primitive.

9.12.3.3 compute_dPdT()

```
Eigen::Matrix<double, 3, -1> DCA::Rectangle::compute_dPdT (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the derivative of a point on the primitive with respect to t.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{dt}$.

In the case of a rectangle, this has size 3x2.

Implements DCA::Primitive.

9.12.3.4 compute_d2PdS2()

```
std::array<Eigen::Matrix<double, 3, 6>, 6> DCA::Rectangle::compute_d2PdS2 (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the second derivative of a point on the primitive with respect to s.

Compute the second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The second derivative $\frac{d^2P}{ds^2}$ as an array (size 6).

Implements DCA::Primitive.

9.12.3.5 compute_d2PdSdT()

```
std::vector<Eigen::Matrix<double, 3, 6> > DCA::Rectangle::compute_d2PdSdT (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the mixed second derivative of a point on the primitive with respect to s and t.

Compute the mixed second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{d^2P}{dsdt}$ as a vector (size t).

In the case of a rectangle, this returns a vector with size 2.

Implements DCA::Primitive.

9.12.3.6 get_s()

```
Vector6d DCA::Rectangle::get_s ( ) const  [override], [virtual]
```

Helper to get the state.

Returns

The state of the primitive.

Implements DCA::Primitive.

9.12.3.7 SIZE_T()

```
int DCA::Rectangle::SIZE_T ( ) const  [override], [virtual]
```

Helper to get the size of t.

Returns

The number of dimensions needed for parameterization.

In the case of a rectangle, this returns 2.

Implements DCA::Primitive.

9.12.3.8 get_largest_dimension_from_center()

```
double DCA::Rectangle::get_largest_dimension_from_center ( ) const  [override], [virtual]
```

Get the largest dimension.

Helper for pair generation.

Returns

The largest dimension from the center point.

Implements DCA::Primitive.

9.12.3.9 getLocalVectors()

```
std::pair<Vector3d, Vector3d> DCA::Rectangle::getLocalVectors ( ) const  [private]
```

Get vectors from the coordinates.

Returns

The local vectors.

9.12.4 Member Data Documentation

9.12.4.1 center

`Vector3d DCA::Rectangle::center`

The center of mass of the rectangle.

9.12.4.2 orientation

`Matrix3d DCA::Rectangle::orientation`

The orientation of the rectangle.

9.12.4.3 dimensions

`Vector2d DCA::Rectangle::dimensions`

Convention: local dimensions in x and z direction.

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

- Primitives.h

9.13 DCA::SoftUnilateralConstraint Class Reference

This class is a soft constraint, meaning a constraint is softified.

`#include <SoftUnilateralConstraint.h>`

Public Member Functions

- SoftUnilateralConstraint (double limit, double stiffness, double epsilon)
  Create a constraint.
- ∼SoftUnilateralConstraint ()=default
  Default deconstructor.
- void setLimit (double limit)
  Set a new limit.
- void setEpsilon (double eps)
  Set a new epsilon.
- void setStiffness (double s)
  Set a new stiffness.
- double evaluate (double x) const
  Compute the force acting on x.
- double computeDerivative (double x) const
  Compute first derivative of the force acting on x.
- double computeSecondDerivative (double x) const
  Compute the second derivative of the force acting on x.

Private Attributes

- double a1
- double b1
- double c1
- double a2
- double b2
- double c2
- double d2
- double epsilon
- double limit

  Internal values.

9.13.1 Detailed Description

This class is a soft constraint, meaning a constraint is softified.

used to model unilateral constraints of the type x $<$ u using a C2 penalty energy f(x).

- u is the upper limit that x needs to be less than
- if x $<$ u, then the energy of the constraint, its gradient and hessian are all 0 (i.e. inactive)
- epsilon is the value away from the limit (how much smaller should x be compared to u) after which f(x) = 0
- stiffness controls the rate at which f(x) increases if x $>$ u

9.13.2 Constructor & Destructor Documentation

9.13.2.1 SoftUnilateralConstraint()

```
DCA::SoftUnilateralConstraint::SoftUnilateralConstraint (
            double limit,
            double stiffness,
            double epsilon )
```

Create a constraint.

Parameters

in	*limit*	The limit of the constraint.
in	*stiffness*	The stiffness of the softifying.
in	*epsilon*	The limit (how close to the limit should the value be able to be).

9.13.2.2 ∼SoftUnilateralConstraint()

```
DCA::SoftUnilateralConstraint::~SoftUnilateralConstraint ( )  [default]
```

Default deconstructor.

9.13.3 Member Function Documentation

9.13.3.1 setLimit()

```
void DCA::SoftUnilateralConstraint::setLimit (
            double limit )
```

Set a new limit.

Parameters

in	*limit*	The new limit.

9.13.3.2 setEpsilon()

```
void DCA::SoftUnilateralConstraint::setEpsilon (
            double eps )
```

Set a new epsilon.

Parameters

in	*eps*	The new epsilon.

9.13.3.3 setStiffness()

```
void DCA::SoftUnilateralConstraint::setStiffness (
            double s )
```

Set a new stiffness.

Parameters

in	*s*	The new stiffness.

9.13.3.4 evaluate()

```
double DCA::SoftUnilateralConstraint::evaluate (
            double x ) const
```

Compute the force acting on x.

Parameters

in	*x*	The current position

Returns

The force F acting on x.

9.13.3.5 computeDerivative()

```
double DCA::SoftUnilateralConstraint::computeDerivative (
            double x ) const
```

Compute first derivative of the force acting on x.

Parameters

in	*x*	The current position

Returns

The first derivative of F evaluated at x.

9.13.3.6 computeSecondDerivative()

```
double DCA::SoftUnilateralConstraint::computeSecondDerivative (
            double x ) const
```

Compute the second derivative of the force acting on x.

Parameters

in	*x*	The current position

Returns

The second derivative of F evaluated at x.

9.13.4 Member Data Documentation

9.13.4.1 a1

`double DCA::SoftUnilateralConstraint::a1 [private]`

9.13.4.2 b1

`double DCA::SoftUnilateralConstraint::b1 [private]`

9.13.4.3 c1

`double DCA::SoftUnilateralConstraint::c1 [private]`

9.13.4.4 a2

`double DCA::SoftUnilateralConstraint::a2 [private]`

9.13.4.5 b2

`double DCA::SoftUnilateralConstraint::b2 [private]`

9.13.4.6 c2

`double DCA::SoftUnilateralConstraint::c2 [private]`

9.13.4.7 d2

```
double DCA::SoftUnilateralConstraint::d2  [private]
```

9.13.4.8 epsilon

```
double DCA::SoftUnilateralConstraint::epsilon  [private]
```

9.13.4.9 limit

```
double DCA::SoftUnilateralConstraint::limit  [private]
```

Internal values.

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

- SoftUnilateralConstraint.h

9.14 DCA::Solver Class Reference

Wrapper around a newton optimizer and an objective.

```
#include <Solver.h>
```

Public Member Functions

- Solver (primitive_t primitive_A, primitive_t primitive_B)

  Create a solver.
- ∼Solver ()=default

  Default deconstructor.
- void compute_t (VectorXd &t, const VectorXd &s, bool forFD=false) const

  Compute the parameterization.
- void compute_dtds (MatrixXd &dtds, const VectorXd &s, const VectorXd &t) const

  Compute the derivative.
- double compute_D (const VectorXd &s, const VectorXd &t) const

  Compute the distance between two primitives.
- void compute_dDdS (VectorXd &dDdS, const VectorXd &s, const VectorXd &t) const

  Compute the derivative of the distance between two primitives w.r.t. s.
- void compute_d2DdS2 (MatrixXd &d2DdS2, const VectorXd &s, const VectorXd &t) const

  Compute the second derivative of the distance between two primitives w.r.t. s.
- std::pair< Vector3d, Vector3d > compute_closest_points (const VectorXd &s, const VectorXd &t) const

  Compute the closest points which yield the shortest distance.
- bool are_t_and_s_in_sync (const VectorXd &s, const VectorXd &t) const

  Check if t and s are in sync.
- Vector12d get_s_from_primitives () const

  Get the state of both primitives, stacked.
- int get_total_size_t () const

  Get the total size of t.

Private Attributes

- Objective objective

  The objective to optimize.
- NewtonOptimizer optimizer

  The newton optimizer.

9.14.1 Detailed Description

Wrapper around a newton optimizer and an objective.

This class actually finds the shortest distance.

9.14.2 Constructor & Destructor Documentation

9.14.2.1 Solver()

```
DCA::Solver::Solver (
            primitive_t primitive_A,
            primitive_t primitive_B )
```

Create a solver.

Parameters

in	*primitive_A*	The first primitive.
in	*primitive_B*	The second primitive.

9.14.2.2 ∼Solver()

```
DCA::Solver::∼Solver ( )  [default]
```

Default deconstructor.

9.14.3 Member Function Documentation

9.14.3.1 compute_t()

```
void DCA::Solver::compute_t (
            VectorXd & t,
            const VectorXd & s,
            bool forFD = false ) const
```

Compute the parameterization.

Parameters

out	*t*	The computed parameterization, stacked.
in	*s*	The state of the two primitives, stacked.
in	*forFD*	Should be used with "false". True will lower residual, which makes it slower but more accurate.

9.14.3.2 compute_dtds()

```
void DCA::Solver::compute_dtds (
            MatrixXd & dtds,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the derivative.

Computes $\frac{dt}{ds}$.

Parameters

out	*dtds*	$\frac{dt}{ds}$.
in	*s*	The state of both primitives, stacked.
in	*t*	The parameterization of both primitives, stacked.

9.14.3.3 compute_D()

```
double DCA::Solver::compute_D (
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the distance between two primitives.

Parameters

in	*s*	The state of the two primitives.
in	*t*	The parameterization of the two primitives, stacked.

Returns

The shortest distance between both primitives, stacked.

9.14.3.4 compute_dDdS()

```
void DCA::Solver::compute_dDdS (
            VectorXd & dDdS,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the derivative of the distance between two primitives w.r.t. s.

Parameters

out	*dDdS*	The derivative $\frac{dD}{ds}$.
in	*s*	The state of the two primitives, stacked.
in	*t*	The parameterization of the two primitives, stacked.

9.14.3.5 compute_d2DdS2()

```
void DCA::Solver::compute_d2DdS2 (
            MatrixXd & d2DdS2,
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the second derivative of the distance between two primitives w.r.t. s.

Parameters

out	*d2DdS2*	The derivative $\frac{d^2D}{ds^2}$.
in	*s*	The state of the two primitives, stacked.
in	*t*	The parameterization of the two primitives, stacked.

9.14.3.6 compute_closest_points()

```
std::pair<Vector3d, Vector3d> DCA::Solver::compute_closest_points (
            const VectorXd & s,
            const VectorXd & t ) const
```

Compute the closest points which yield the shortest distance.

Parameters

in	*s*	The state of the two primitives, stacked.
in	*t*	The parameterization of the two primitives, stacked.

Returns

A pair of points, where the distance between those points is the shortest between the two primitives.

9.14.3.7 are_t_and_s_in_sync()

```
bool DCA::Solver::are_t_and_s_in_sync (
            const VectorXd & s,
            const VectorXd & t ) const
```

Check if t and s are in sync.

Parameters

in	*s*	The state of the two primitives, stacked.
in	*t*	The parameterization of the two primitives, stacked.

Returns

True, if t and s are in sync. False, otherwise.

9.14.3.8 get_s_from_primitives()

```
Vector12d DCA::Solver::get_s_from_primitives ( ) const
```

Get the state of both primitives, stacked.

Returns

The state of the two primitives, stacked.

9.14.3.9 get_total_size_t()

```
int DCA::Solver::get_total_size_t ( ) const
```

Get the total size of t.

Returns

The total size of t.

9.14.4 Member Data Documentation

9.14.4.1 objective

`Objective DCA::Solver::objective [private]`

The objective to optimize.

9.14.4.2 optimizer

`NewtonOptimizer DCA::Solver::optimizer [mutable], [private]`

The newton optimizer.

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

- Solver.h

9.15 DCA::Sphere Class Reference

Definition of a Sphere.

`#include <Primitives.h>`

Public Member Functions

- Sphere (const Vector3d &position, const double &radius)
- ∼Sphere ()=default

    Default deconstructor.
- Vector3d compute_P (const Vector6d &s, const VectorXd &t) const override

    Compute a point on the primitive.
- Eigen::Matrix< double, 3, 6 > compute_dPdS (const Vector6d &s, const VectorXd &t) const override

    Compute the derivative of a point on the primitive with respect to s.
- Eigen::Matrix< double, 3, -1 > compute_dPdT (const Vector6d &s, const VectorXd &t) const override

    Compute the derivative of a point on the primitive with respect to t.
- std::array< Eigen::Matrix< double, 3, 6 >, 6 > compute_d2PdS2 (const Vector6d &s, const VectorXd &t) const override

    Compute the second derivative of a point on the primitive with respect to s.
- std::vector< Eigen::Matrix< double, 3, 6 > > compute_d2PdSdT (const Vector6d &s, const VectorXd &t) const override

    Compute the mixed second derivative of a point on the primitive with respect to s and t.
- Vector6d get_s () const override

    Helper to get the state.
- int SIZE_T () const override

    Helper to get the size of t.
- double get_largest_dimension_from_center () const override

    Get the largest dimension.

Public Attributes

- Vector3d position

    The position of the sphere in 3d space.

Additional Inherited Members

9.15.1 Detailed Description

Definition of a Sphere.

The state of a sphere is as follows: (x, y, z, -, -, -), which means that it is the center of the sphere and the other three parameters are not used.

The parameterization of a sphere is as follows: (-), which means, it does not need any (0-dimensional).

9.15.2 Constructor & Destructor Documentation

9.15.2.1 Sphere()

```
DCA::Sphere::Sphere (
            const Vector3d & position,
            const double & radius )
```

Create a sphere primitive.

Parameters

in	*position*	The position of the sphere.
in	*radius*	The safety margin around the center.

9.15.2.2 ∼Sphere()

```
DCA::Sphere::∼Sphere ( ) [default]
```

Default deconstructor.

9.15.3 Member Function Documentation

9.15.3.1 compute_P()

```
Vector3d DCA::Sphere::compute_P (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute a point on the primitive.

Compute a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The point.

The paramterization t is not used, since it has size 0!

Implements DCA::Primitive.

9.15.3.2 compute_dPdS()

```
Eigen::Matrix<double, 3, 6> DCA::Sphere::compute_dPdS (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the derivative of a point on the primitive with respect to s.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{ds}$.

Implements DCA::Primitive.

9.15.3.3 compute_dPdT()

```
Eigen::Matrix<double, 3, -1> DCA::Sphere::compute_dPdT (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the derivative of a point on the primitive with respect to t.

Compute the derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{dP}{dt}$.

In the case of a sphere, this has size 3x0.

Implements DCA::Primitive.

9.15.3.4 compute_d2PdS2()

```
std::array<Eigen::Matrix<double, 3, 6>, 6> DCA::Sphere::compute_d2PdS2 (
            const Vector6d & s,
            const VectorXd & t ) const  [override], [virtual]
```

Compute the second derivative of a point on the primitive with respect to s.

Compute the second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The second derivative $\frac{d^2P}{ds^2}$ as an array (size 6).

Implements DCA::Primitive.

9.15.3.5 compute_d2PdSdT()

```
std::vector<Eigen::Matrix<double, 3, 6> > DCA::Sphere::compute_d2PdSdT (
            const Vector6d & s,
            const VectorXd & t ) const [override], [virtual]
```

Compute the mixed second derivative of a point on the primitive with respect to s and t.

Compute the mixed second derivative of a point on the primitive based on s and t.

Parameters

in	*s*	The state of the primitive.
in	*t*	The parameterization of the point.

Returns

The derivative $\frac{d^2P}{dsdt}$ as a vector (size t).

In the case of a sphere, this returns an empty vector.

Implements DCA::Primitive.

9.15.3.6 get_s()

```
Vector6d DCA::Sphere::get_s ( ) const [override], [virtual]
```

Helper to get the state.

Returns

The state of the primitive.

Implements DCA::Primitive.

9.15.3.7 SIZE_T()

```
int DCA::Sphere::SIZE_T ( ) const [override], [virtual]
```

Helper to get the size of t.

Returns

The number of dimensions needed for parameterization.

In the case of a sphere, this returns 0.

Implements DCA::Primitive.

9.15.3.8 get_largest_dimension_from_center()

```
double DCA::Sphere::get_largest_dimension_from_center ( ) const  [override], [virtual]
```

Get the largest dimension.

Helper for pair generation.

Returns

The largest dimension from the center point.

In the case of a sphere, this returns the radius.

Implements DCA::Primitive.

9.15.4 Member Data Documentation

9.15.4.1 position

```
Vector3d DCA::Sphere::position
```

The position of the sphere in 3d space.

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

- Primitives.h

Chapter 10

File Documentation

10.1 API.h File Reference

```
#include <DCA/Pair.h>
```

Classes

- class DCA::API
  Public API.

Namespaces

- DCA

10.2 ddR.h File Reference

```
#include <Eigen/Core>
```

Namespaces

- DCA

Functions

- Eigen::Matrix3d DCA::ddR_0_0 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_0 d\theta_0}$.
- Eigen::Matrix3d DCA::ddR_0_1 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_0 d\theta_1}$.
- Eigen::Matrix3d DCA::ddR_0_2 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_0 d\theta_2}$.
- Eigen::Matrix3d DCA::ddR_1_0 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_1 d\theta_0}$.
- Eigen::Matrix3d DCA::ddR_1_1 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_1 d\theta_1}$.
- Eigen::Matrix3d DCA::ddR_1_2 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_1 d\theta_2}$.
- Eigen::Matrix3d DCA::ddR_2_0 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_2 d\theta_0}$.
- Eigen::Matrix3d DCA::ddR_2_1 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_2 d\theta_1}$.
- Eigen::Matrix3d DCA::ddR_2_2 (const Eigen::Vector3d &v)

  Compute $\frac{d^2R}{d\theta_2 d\theta_2}$.

10.3 ExpCoords.h File Reference

```
#include <DCA/Logger.h>
#include <DCA/Utils.h>
#include <DCA/ddR.h>
```

Classes

- class DCA::ExpCoords

  Represents Rotations using exponential coordinates.

Namespaces

- DCA

10.4 FiniteDifference.h File Reference

```
#include <DCA/Utils.h>
#include <functional>
```

Classes

- class DCA::FiniteDifference

  Helper for finite differences (to test derivatives).

Namespaces

- DCA

10.5 Logger.h File Reference

```
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <vector>
```

Classes

- class DCA::Logger

  Logger class.

Namespaces

- DCA

10.6 NewtonOptimizer.h File Reference

```
#include <DCA/Objective.h>
```

Classes

- class DCA::NewtonOptimizer

  Simply newton optimizer class.

Namespaces

- DCA

10.7 Objective.h File Reference

```
#include <DCA/FiniteDifference.h>
#include <DCA/Primitives.h>
#include <DCA/SoftUnilateralConstraint.h>
```

Classes

- class DCA::Objective

  The main objective which used for computing distances and their derivatives.

Namespaces

- DCA

10.8 Pair.h File Reference

```
#include <DCA/Primitives.h>
```

Classes

- class DCA::PermutationPairGenerator

  This generator creates all possible permutations of pairs. This means, the amount of pairs created is $n^2/2$, where n is the number of primitives. Pairs are not returned twice (0, 1) and (1, 0).
- class DCA::NeighborsPairGenerator

  This generator computes the pairs which are in a certain threshold from each other. It does so by computing a single position for each primitive and selecting pairs based on the distance.

Namespaces

- DCA

10.9 Primitives.h File Reference

```
#include <DCA/FiniteDifference.h>
#include <DCA/Utils.h>
#include <array>
#include <variant>
#include <vector>
```

Classes

- class DCA::Primitive

  Definition of a Primitive.
- class DCA::Sphere

  Definition of a Sphere.
- class DCA::Capsule

  Definition of a Capsule.
- class DCA::Rectangle

  Definition of a Rectangle.
- class DCA::Box

  Definition of a Box.

Namespaces

- DCA

Typedefs

- using DCA::primitive_t = std::variant< Sphere, Capsule, Rectangle, Box >

  All possible primitives.

10.10 README.md File Reference

10.11 SoftUnilateralConstraint.h File Reference

Classes

- class DCA::SoftUnilateralConstraint

  This class is a soft constraint, meaning a constraint is softified.

Namespaces

- DCA

10.12 Solver.h File Reference

```
#include <DCA/NewtonOptimizer.h>
#include <DCA/Objective.h>
```

Classes

- class DCA::Solver

  Wrapper around a newton optimizer and an objective.

Namespaces

- DCA

10.13 Utils.h File Reference

```
#include <Eigen/Core>
#include <Eigen/Geometry>
#include <vector>
```

Namespaces

- DCA

Typedefs

- typedef unsigned int DCA::uint

  Easier access to a unsigned int.
- using DCA::Vector0d = Eigen::Matrix< double, 0, 1 >

  Easier access to a 0 vector of type double.
- using DCA::Vector1d = Eigen::Matrix< double, 1, 1 >

  Easier access to a 1 vector of type double.
- using DCA::Vector6d = Eigen::Matrix< double, 6, 1 >

  Easier access to a 6 vector of type double.
- using DCA::Vector12d = Eigen::Matrix< double, 12, 1 >

  Easier access to a 12 vector.
- using DCA::Matrix12d = Eigen::Matrix< double, 12, 12 >

  Easier access to a 12 by 12 matrix.
- using DCA::pair_t = std::pair< size_t, size_t >

  Easier access to a pair (corresponding of two indices)

Index