Root Tracking

Generated by Doxygen 1.8.17

Chapter 1

Todo List

Member linearSolve (MatrixXd Amat, VectorXd bvec)

Add assertions for square matrix verification and verification of sizes of A and b

Class RootTracker

See how things change when openMP is enabled with Eigen

See how things change when BLAS and LAPACK are used with Eigen

Implement an event identification method that can handle or atleast alert when the system moves close to a singularity

modify the paper with a definition of distance as all the individual elements of the vector being within the eps radius ball. So once we define the eps, we take 2 steps, which gives us 4 solution values, i.e. two branches two sols. Then use a multivariable interpolation scheme to interpolate from both sides. Then find their intersection to find the location of singularity.

Member RootTracker::DMTracker (VectorXd xprev, VectorXd x, VectorXd y, std::function< MatrixXd(VectorXd)> Jfx, std::function< MatrixXd(VectorXd)> Jfy, double eps=0, std::function< VectorXd(VectorXd)> f=NULL)

Provide support for different integration methods

This isnt working. Implement it using prevtheta like in Mathematica rather than using xnext. That should fix it.

The same scheme estimating dx as xnext-x is unstable

Member RootTracker::Methods ()

Give provision for an FK solver along with root trackers to use when trackers fail.

Member RootTracker::NNTracker (VectorXd ys, MatrixXd ysols, int index)

The method currently deals only with variables belonging to R and S1.

Treating the Rodriques parameters as locally belonging to R.

Add assertions for cols of ys and ysols to be same.

Member RootTracker::NRTracker (VectorXd x, VectorXd y, std::function< VectorXd(VectorXd)> f, std::function< MatrixXd(VectorXd)> Jfy, double eps=pow(10, -10))

Decide to return q or y in the NRTracker method

fval returning nan is not being handled in the while loop

Input argument format and other relevant checks

Member RootTracker::SingularityEventIdentifier (VectorXd ys, MatrixXd ysols, double eps=pow(10, -2))

Integrate Bertini to find all the roots

Member s1Dist (VectorXd theta1, VectorXd theta2)

Can do more effecient computation if the angles mapping is saved in the memory

Assert sizes of theta1 and theta2 to be same

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 RootTracker Class Reference

```
#include <root_tracker.hh>
```

Public Member Functions

- VectorXd NRTracker (VectorXd x, VectorXd y, std::function< VectorXd(VectorXd)> f, std::function< MatrixXd(VectorXd)> Jfy, double eps=pow(10, -10))
- VectorXd DMTracker (VectorXd xprev, VectorXd x, VectorXd y, std::function< MatrixXd(VectorXd)> Jfx, std::function< MatrixXd(VectorXd)> Jfy, double eps=0, std::function< VectorXd(VectorXd)> f=NULL)
- VectorXd NNTracker (VectorXd ys, MatrixXd ysols, int index)
- int Methods ()
- VectorXd SingularityEventIdentifier (VectorXd ys, MatrixXd ysols, double eps=pow(10, -2))

4.1.1 Detailed Description

The RootTracker class consists of implementations of the following:

- Newton-Raphson method based tracking or NRTracker
- Davidenkos method based tracking or DMTracker
- Nearest neighbour based tracking or NNTracker

Todo See how things change when openMP is enabled with Eigen

See how things change when BLAS and LAPACK are used with Eigen

Implement an event identification method that can handle or atleast alert when the system moves close to a singularity

Todo modify the paper with a definition of distance as all the individual elements of the vector being within the eps radius ball. So once we define the eps, we take 2 steps, which gives us 4 solution values, i.e. two branches two sols. Then use a multivariable interpolation scheme to interpolate from both sides. Then find their intersection to find the location of singularity.

4.1.2 Member Function Documentation

4.1.2.1 DMTracker()

```
VectorXd RootTracker::DMTracker (
            VectorXd xprev,
            VectorXd x,
            VectorXd y,
            std::function< MatrixXd(VectorXd)> Jfx,
            std::function< MatrixXd(VectorXd)> Jfy,
            double eps = 0,
            std::function< VectorXd(VectorXd)> f = NULL )
```

The DMTracker uses the Davindenkos' integration method to find the solutions satisfied by the constrain equations. The output is the values of the unknown variables at each tracking step. The problem of tracking is solved as an initial value problem using the first order derivative form of the constraint equations. This function accepts functions as arguments using the `C++11` style functional library. This method by default uses the Explicit Euler integration scheme and is the only supported scheme currently.

Parameters

xprev	The set of input/known variables at the previous tracking step
x	The set of input/known variables at the current tracking step
y	The set of output/unknown variables at current step
Jfx	The expression of the Jacobian matrix of f with respect to x, the known variables. Takes in a single argument of type `VectorXd` consisting y, x and outputs the evaluation of Jfx
Jfy	The expression of the Jacobian matrix of f with respect to y, the unknown variables. Takes in a single argument of type `VectorXd` consisting y, x and outputs the evaluation of Jfy.
eps	The tolerance of `drift` to which the computed solutions are to satisfy the non-linear equations. If the drift exceeds the given tolerance, a Newton-Raphson (NR) step is used to bring the variables back to the constraint manifold. The default value of `eps` is set to 0, meaning the NR step correction is `off` by default.
f	The set of expressions of the non-linear functions relating x, y. This parameter is only required if `eps` is defined.

Todo Provide support for different integration methods

This isnt working. Implement it using prevtheta like in Mathematica rather than using xnext. That should fix it.

The same scheme estimating dx as xnext-x is unstable

4.1.2.2 Methods()

```
int RootTracker::Methods ( )
```

The methods function can be called to view names of all the implemented root tracking methods.

Todo Give provision for an FK solver along with root trackers to use when trackers fail.

4.1.2.3 NNTracker()

```
VectorXd RootTracker::NNTracker (
            VectorXd ys,
            MatrixXd ysols,
            int index )
```

The NNTracker uses the nearest neighbour method to identify the roots belonging to a required branch. The output is the selected root at each tracking step. This method assumes the existance of a solver method, `Solve`, which computes all the roots for given input variables, `x`. The function expects the solutions to be ordered with all the reals together and the variables belonging to S1 together.

Parameters

ys	The initiation of the known root of the required branch
ysols	All the solutions obtained by the `Solve` method used
index	The index upto which the reals are present.

Todo The method currently deals only with variables belonging to R and S1.

Treating the Rodriques parameters as locally belonging to R.

Add assertions for cols of ys and ysols to be same.

4.1.2.4 NRTracker()

```
VectorXd RootTracker::NRTracker (
            VectorXd x,
            VectorXd y,
            std::function< VectorXd(VectorXd)> f,
            std::function< MatrixXd(VectorXd)> Jfy,
            double eps = pow(10, -10) )
```

The NRTracker uses the Newton-Raphson method iteratively to find the solutions satisfied by the constrain equations. The output is the values of the unknown variables at each tracking step. This function accepts functions as arguments using the `C++11` style functional library.

Parameters

x	The set of input/known variables at current tracking step
y	The set of output/unknown variables at current step
f	The set of expressions of the non-linear functions relating x, y. Takes in a single argument of type `VectorXd` consisting y, x and outputs the evaluation of f.
Jfy	The expression of the Jacobian matrix of f with respect to y, the unknown variables. Takes in a single argument of type `VectorXd` consisting y, x and outputs the evaluation of Jfy.
eps	The required tolerance to which the computed solutions are to satisfy the non-linear equations. Default value is set to 10^-10

Todo Decide to return q or y in the NRTracker method fval returning nan is not being handled in the while loop

Input argument format and other relevant checks

4.1.2.5 SingularityEventIdentifier()

```
VectorXd RootTracker::SingularityEventIdentifier (
            VectorXd ys,
            MatrixXd ysols,
            double eps = pow(10, -2) )
```

The SingularityEventIdentifier uses a distance metric to identify when the configuration approaches a singularity. Further, it uses a linear interpolation to estimate the singular configuration. This function needs the computation of all the roots, real or imaginary to be provided. Optionally Bertini can be used to compute all the roots.

Parameters

ys	The current root of the required branch
ysols	All the roots at the instant
eps	The distance tolerance after which the singularity event is triggered

Todo Integrate Bertini to find all the roots

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

- root_tracker.hh
- root_tracker.cc

Chapter 5

File Documentation

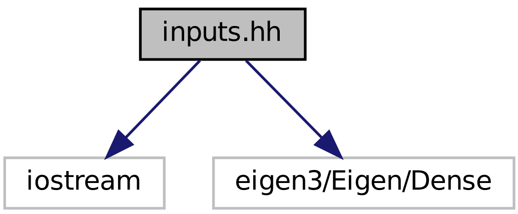
5.1 DMTracker_noNR.txt File Reference

5.2 DMTracker_NR.txt File Reference

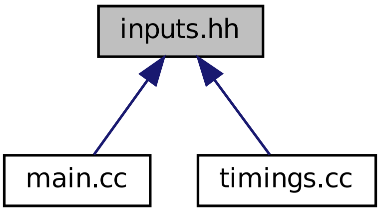
5.3 inputs.hh File Reference

```
#include <iostream>
#include <eigen3/Eigen/Dense>
```

Include dependency graph for inputs.hh:



This graph shows which files directly or indirectly include this file:



Functions

- VectorXd etaext (VectorXd q)
- MatrixXd Jetaexttheta (VectorXd q)
- MatrixXd Jetaextphi (VectorXd q)
- Matrix< double, 51, 6 > legvals (legvals_data)

Variables

- const double legvals_data [ ]

5.3.1 Function Documentation

5.3.1.1 etaext()

```
VectorXd etaext (
            VectorXd q )
```

The funcion etaext is the set of constraint equations of an SRSPM formulated in the extended configuration space of the manipulator. This is for the example problem demosnstrating the use of the root-trackers. The formulation of the symbolic equations can be found in the accompanying Mathematica notebook named, `SRSPM_root_tracking.nb`.

Parameters

q	Takes in the 24-dimensional extended-configuration-space variables as input

5.3.1.2 Jetaextphi()

```
MatrixXd Jetaextphi (
            VectorXd q )
```

The funcion Jetaextphi is the constraint Jacobian matrix of an SRSPM formulated in the extended configuration space of the manipulator. This is for the example problem demonstrating the use of the root-trackers. The formulation of the symbolic equations can be found in the accompanying Mathematica notebook named, `SRSPM_root_tracking.nb`.

Parameters

q	Takes in the 24-dimensional extended-configuration-space variables as input

5.3.1.3 Jetaexttheta()

```
MatrixXd Jetaexttheta (
            VectorXd q )
```

The funcion Jetaexttheta is the Jacobian matrix of an SRSPM formulated in the extended configuration space of the manipulator. This is for the example problem demonstrating the use of the root-trackers. The formulation of the symbolic equations can be found in the accompanying Mathematica notebook named, `SRSPM_root_tracking.nb`.

Parameters

q	Takes in the 24-dimensional extended-configuration-space variables as input

5.3.1.4 legvals()

```
Matrix<double, 51, 6> legvals (
            legvals_data )
```

Leg values corresponding to the path to be tracked in the task-space typecasted into a matrix

5.3.2 Variable Documentation

5.3.2.1 legvals_data

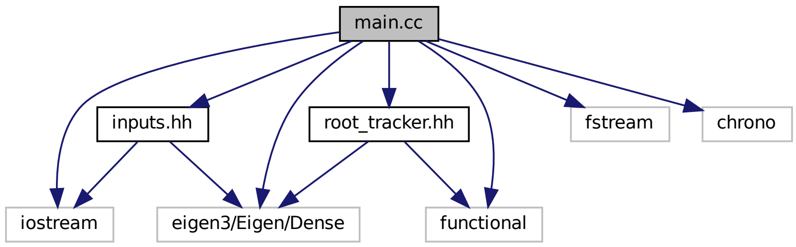
```
const double legvals_data[ ]
```

An array of leg values corresponding to the path to be tracked in the task-space

5.4 main.cc File Reference

```
#include <iostream>
#include <eigen3/Eigen/Dense>
#include "root_tracker.hh"
#include "inputs.hh"
#include <fstream>
#include <functional>
#include <chrono>
```

Include dependency graph for main.cc:



Functions

- void saveData (MatrixXd A, const char file_name[ ])
- int main (int argc, char const ∗argv[ ])

5.4.1 Function Documentation

5.4.1.1 main()

```
int main (
        int argc,
        char const * argv[ ] )
```

Example problem demonstrating the usage of the root trackers for solving a task space path following problem of an SRSPM using all the discussed root-trackers. Choosing small enough `eps` makes DMTracker work like an NRTracker

5.4.1.2 saveData()

```
void saveData (
        MatrixXd A,
        const char file_name[ ] )
```

The saveData function takes in an input `MatrixXd` and saves it in a .txt file

Parameters

A	Input matrix
file_name	Name of the file to save the input matrix in

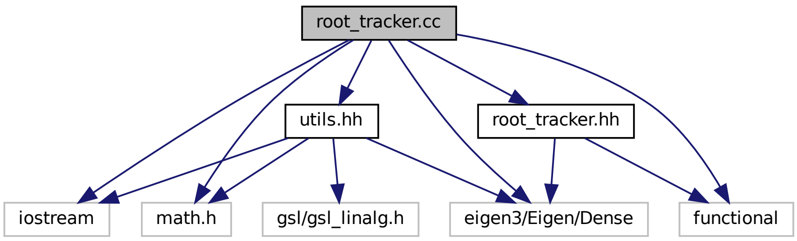
5.5 NNTracker_NR.txt File Reference

5.6 NRTracker.txt File Reference

5.7 root_tracker.cc File Reference

```
#include <iostream>
#include <math.h>
#include <eigen3/Eigen/Dense>
#include "root_tracker.hh"
#include "utils.hh"
#include <functional>
```

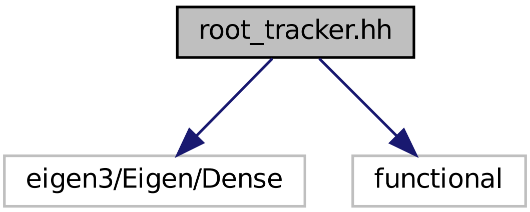
Include dependency graph for root_tracker.cc:



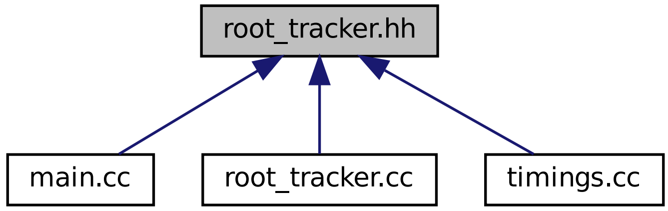
5.8 root_tracker.hh File Reference

```
#include <eigen3/Eigen/Dense>
#include <functional>
```

Include dependency graph for root_tracker.hh:



This graph shows which files directly or indirectly include this file:



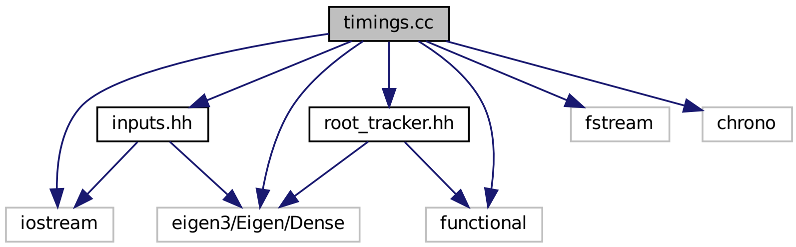
Classes

- class RootTracker

5.9 timings.cc File Reference

```
#include <iostream>
#include <eigen3/Eigen/Dense>
#include "root_tracker.hh"
#include "inputs.hh"
#include <fstream>
#include <functional>
#include <chrono>
```

Include dependency graph for timings.cc:



Functions

- void saveData (MatrixXd A, const char file_name[ ])
- int main (int argc, char const ∗argv[ ])

5.9.1 Function Documentation

5.9.1.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

This file is the same as the `main.cc` file. But is used to compute the times taken by each of the methods over 1000 runs. Choosing small enough `eps` makes DMTracker work like an NRTracker

5.9.1.2 saveData()

```
void saveData (
            MatrixXd A,
            const char file_name[] )
```

The saveData function takes in an input `MatrixXd` and saves it in a .txt file

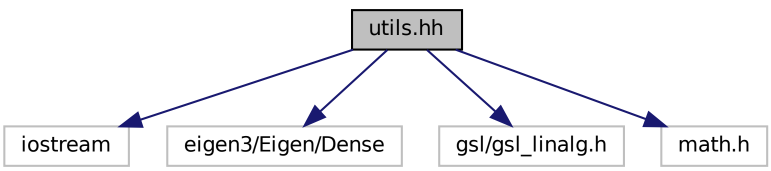
Parameters

A	Input matrix
file_name	Name of the file to save the input matrix in

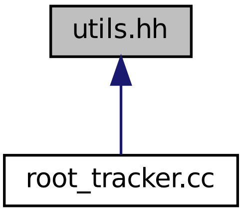
5.10 utils.hh File Reference

```
#include <iostream>
#include <eigen3/Eigen/Dense>
#include <gsl/gsl_linalg.h>
#include <math.h>
```

Include dependency graph for utils.hh:



This graph shows which files directly or indirectly include this file:



Functions

- VectorXd linearSolve (MatrixXd Amat, VectorXd bvec)
- VectorXd s1Dist (VectorXd theta1, VectorXd theta2)

5.10.1 Function Documentation

5.10.1.1 linearSolve()

```
VectorXd linearSolve (
            MatrixXd Amat,
            VectorXd bvec )
```

The linearSolve function is a simple, usable wraper around the GSLs linear solver using the Eigen library.

Parameters

Amat	Input matrix
bvec	Input vector

Todo Add assertions for square matrix verification and verification of sizes of A and b

5.10.1.2 s1Dist()

```
VectorXd s1Dist (
            VectorXd theta1,
            VectorXd theta2 )
```

The s1Dist returns the distance between two elements in the S1 space.

Parameters

theta1	First element
theta2	Second element

Todo Can do more effecient computation if the angles mapping is saved in the memory

Assert sizes of theta1 and theta2 to be same

Index