My Project

Generated by Doxygen 1.9.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

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 documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Aorta Class Reference

Public Member Functions

- bool s_point (Eigen::Matrix4d point)

  Adds a single point to the point cloud of the Aorta.
- bool s_points (Eigen::MatrixXd points)

  Adds a group of points to the point cloud of the Aorta.
- bool s_deadD (double deadD)

  Sets the dead distance for the Aorta.
- bool **s_dangerD** (double dangerD)
- bool **s_safeD** (double safeD)
- void **clear** ()
- void **checkDistance** (Eigen::Matrix4d ee)

  does somthegn
- int g_safety ()

  Returns the calculated safety value of the current simulation.
- double **g_maxDist** ()
- double g_deadD ()

  Returns the dead distance.
- double g_dangerD ()

  Returns the danger distance.
- Eigen::MatrixXd **g_points** ()

  Returns all points set by s_point() and s_points()

4.1.1 Member Function Documentation

4.1.1.1 g_dangerD()

```
double Aorta::g_dangerD ( )
```

Returns the danger distance.

Parameters

out		

verbatim Returns -1 if s_dangerD() has not yet been called Returns dangerD otherwise

4.1.1.2 g_deadD()

```
double Aorta::g_deadD ( )
```

Returns the dead distance.

Parameters

out		

verbatim Returns -1 if s_deadD() has not yet returned true. Returns deadD otherwise

4.1.1.3 g_safety()

```
int Aorta::g_safety ( )
```

Returns the calculated safety value of the current simulation.

Calculates a safety value from a single point and the every part of the point cloud. The safety value is based on the distance to the cloud

Parameters

out	*int;*	ranging from -1 to 2 `-1 if checkDistance() has not yet been run and s_deadD() and s_dangerD() have both not yet retu` `0 if the distance is less than the Dead Distance` `1 if the distance is greater than Dead Distance but less than Danger Distance` `2 if the distance is greater than Danger Distance`

4.1.1.4 s_deadD()

```
bool Aorta::s_deadD (
             double deadD )
```

Sets the dead distance for the Aorta.

The minimum distance something can come to the aorta.

Parameters

in	*deadD*	

Parameters

out	*True/False*	`True if the distance is within the predefined acceptable range.`
		`False if it falls outside the predefined acceptable range.`

4.1.1.5 s_point()

```
bool Aorta::s_point (
             Eigen::Matrix4d point )
```

Adds a single point to the point cloud of the Aorta.

Parameters

in	*point*	

out	*True/False*	`True if the point does not already exist.`
		`False if the point does already exist.`

4.1.1.6 s_points()

```
bool Aorta::s_points (
             Eigen::MatrixXd points )
```

Adds a group of points to the point cloud of the Aorta.

Parameters

in	*points*	

out	*True/False*	`True if all the points did not yet exist.`
		`False if any of the points did already exist.`

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

- include/aorta.h
- src/aorta.cpp

4.2 Catheter Class Reference

Public Member Functions

- bool **s_baseFrame** (Eigen::Matrix4d baseFrame)

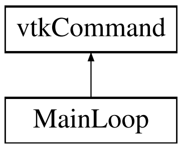
- bool **s_nseg** (int nseg)
- bool **s_nq** (int nq)
- bool **s_pps** (int pps)
- bool **s_rad** (double rad)
- bool **s_bbLen** (double bbLen)
- void **fkine** (Eigen::MatrixXd q)
- double **g_distEE** ()
- double **g_rad** ()
- double **g_q1change** ()
- double **g_q2change** ()
- double **g_q3change** ()
- double **g_qChange** (int qkind)
- Eigen::MatrixXd **g_q** ()
- Eigen::MatrixXd **g_baseFrame** ()
- Eigen::Matrix4d **g_eeFrame** ()
- Eigen::MatrixXd **g_backbone** ()

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

- include/catheter.h
- src/catheter.cpp

4.3 MainLoop Class Reference

Inheritance diagram for MainLoop:



Public Member Functions

- **MainLoop** (Visualizer vis)
- virtual void **Execute** (vtkObject ∗vtkNotUsed(caller), unsigned long eventId, void ∗vtkNotUsed(callData))

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

- include/mainloop.h
- src/mainloop.cpp

4.4 Visualizer Class Reference

Public Member Functions

- void **drawCath** (Eigen::MatrixXd bb, double rad)
- void **drawAorta** (Eigen::MatrixXd wall, double dead, double danger)
- void **update** ()
- void **clearCath** ()
- void **clearAorta** ()
- void **clear** ()
- vtkSmartPointer< vtkRenderWindow > **g_renderWindow** ()

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

- include/visualizer.h
- src/visualizer.cpp

Chapter 5

File Documentation

5.1 aorta.h

```cpp

// Include
#include <robot.h>

//stl
#include <cmath>
#include <iostream>
#include <ctime>

//Eigen
#include <Eigen/Dense>

class Aorta{
    private:
        double m_deadD;
        double m_dangerD;
        double m_safeD;

        // For Checking
        double m_maxD;
        int m_safety;

        // For Drawing
        Eigen::MatrixXd m_points;

    public:
        // Initalize
        Aorta();
        ~Aorta();

        // Setting Functions
        bool s_point(Eigen::Matrix4d point);
        bool s_points(Eigen::MatrixXd points);
        bool s_deadD(double deadD);
        bool s_dangerD(double dangerD);
        bool s_safeD(double safeD);

        // Doing Functions
        void clear();
        void checkDistance(Eigen::Matrix4d ee);

        // Getting Functions
        int g_safety();
        double g_maxDist();
        double g_deadD();
        double g_dangerD();
        Eigen::MatrixXd g_points();
};
```

5.2 catheter.h

```
#define _USE_MATH_DEFINES
#pragma once

#include "robot.h"

//stl
#include <cmath>
#include <iostream>
#include <ctime>

//Eigen
#include <Eigen/Dense>

// A catheter has the functionality of a TDCR but is drawn as if it is a CTCR
class Catheter
{
    private:
        int m_nseg;
        int m_nq;
        int m_nqps;
        int m_pps;
        double m_rad;
        double m_bbLen;


        double m_q1change;
        double m_q2change;
        double m_q3change;

        //Member variables defining the parameters of the TDCR
        Eigen::MatrixXd m_q;

        // Matrixes
        Eigen::Matrix4d m_baseFrame;
        Eigen::Matrix4d m_eeFrame;
        Eigen::MatrixXd m_backbone;

    public:
        Catheter();
        ~Catheter();

        // Setting Functions
        bool s_baseFrame(Eigen::Matrix4d baseFrame);
        bool s_nseg(int nseg);
        bool s_nq(int nq);
        bool s_pps(int pps);
        bool s_rad(double rad);
        bool s_bbLen(double bbLen);


        // Doing Functions
        void fkine(Eigen::MatrixXd q);


        // Getting Functions
        double g_distEE();
        double g_rad();
        double g_q1change();
        double g_q2change();
        double g_q3change();
        double g_qChange(int qkind);
        Eigen::MatrixXd g_q();
        Eigen::MatrixXd g_baseFrame();
        Eigen::Matrix4d g_eeFrame();
        Eigen::MatrixXd g_backbone();

};
```

5.3 mainloop.h

```
#pragma once

#define _USE_MATH_DEFINES

#include <visualizer.h>
#include <catheter.h>
#include <aorta.h>

//stl
#include <ctime>
#include <cmath>
```

```
#include <fstream>

//vtk
#include <vtkSmartPointer.h>
#include <vtkCommand.h>
#include <vtkRenderWindowInteractor.h>

//Eigen
#include <Eigen/Dense>

// Class that implements the main simulation loop
class MainLoop : public vtkCommand
{
    private:
        Visualizer mp_vis;
        Catheter m_cath;
        Aorta m_aorta;

        Eigen::MatrixXd m_wall;

        bool m_engaged;
        bool m_aortaEn;
        bool m_cathEn;
        Eigen::MatrixXd m_q;
        double m_rotated;
        double m_bended;


    public:
        MainLoop(Visualizer vis);
        ~MainLoop();

        // Execution Function
        virtual void Execute(vtkObject *vtkNotUsed(caller), unsigned long eventId, void
    *vtkNotUsed(callData));
};
```

5.4 robot.h

```
#pragma once

//stl
#include <cmath>
#include <iostream>
#include <ctime>

//Eigen
#include <Eigen/Dense>

Eigen::MatrixXd arc2x(Eigen::Matrix4d baseFrame, Eigen::MatrixXd kappa, Eigen::MatrixXd length,
    Eigen::MatrixXd phi, int n);

// Returns the distance between two different points in space.
double differance(Eigen::Matrix4d p1, Eigen::Matrix4d p2);


//Eigen::MatrixXd arc_to_x(Eigen::Matrix4d init_frame, std::vector<double> kappa, std::vector<double> l,
    std::vector<double> phi, int n, bool bishop);

//Eigen::Matrix4d matrix_log(Eigen::Matrix4d T);


//Eigen::MatrixXd calculate_desired_body_twist(Eigen::Matrix4d T_target, Eigen::Matrix4d T_cur);

```

5.5 visualizer.h

```
#pragma once

#define _USE_MATH_DEFINES

//stl
#include <vector>
#include <array>

//vtk
#include <vtkRenderer.h>
#include <vtkRenderWindow.h>
#include <vtkSmartPointer.h>
```

```
#include <vtkActor.h>
#include <vtkAxesActor.h>
#include <vtkPolyData.h>
#include <vtkPolyDataMapper.h>
#include <vtkProperty.h>
#include <vtkTransform.h>
#include <vtkCellArray.h>
#include <vtkOpenGLLight.h>
#include <vtkUnstructuredGrid.h>
#include <vtkGeometryFilter.h>
#include <vtkMatrix4x4.h>
#include <vtkCamera.h>
#include <vtkTubeFilter.h>

// Shapes
#include <vtkLine.h>
#include <vtkLineSource.h>
#include <vtkTriangle.h>
#include <vtkCubeSource.h>
#include <vtkSphereSource.h>
#include <vtkCylinderSource.h>

#include <vtkNamedColors.h>
#include <vtkRenderWindowInteractor.h>

//Eigen
#include <Eigen/Dense>

// Class that implements the visualizer of the simulator using VTK
class Visualizer{
    private:
        vtkSmartPointer<vtkRenderer> mp_Ren;
        vtkSmartPointer<vtkRenderWindow> mp_RenWin;
        vtkSmartPointer<vtkAxesActor> mp_target_frame;
        vtkSmartPointer<vtkNamedColors> mp_colors; // This is giving me trouble look into it later
        //std::vector<vtkSmartPointer<vtkActor» mp_curves;
        std::vector<vtkSmartPointer<vtkActor» m_curveActors;
        std::vector<vtkSmartPointer<vtkActor» m_sphereActors;
        std::vector<vtkSmartPointer<vtkActor» m_frameActors;
        std::vector<vtkSmartPointer<vtkActor» m_cathActors;
        std::vector<vtkSmartPointer<vtkActor» m_aortaActors;

        // Drawing Functions
        void drawCurves(Eigen::MatrixXd curve, double rad);
        void drawFrames(Eigen::MatrixXd frames);
        void drawPoints(Eigen::MatrixXd points, double rad, char color);
        void drawSphere(Eigen::MatrixXd points, std::vector<vtkSmartPointer<vtkActor» &actors, double
    rad, std::vector<double> color, double trans);

    public:
        // Init functions
        Visualizer();
        ~Visualizer();

        // Set Functions

        // Do functions
            // Draw
        void drawCath(Eigen::MatrixXd bb, double rad);
        void drawAorta(Eigen::MatrixXd wall, double dead, double danger);
        void update();

            // Clear
        void clearCath();
        void clearAorta();
        void clear();

        // Get Functions
        vtkSmartPointer<vtkRenderWindow> g_renderWindow();
};
```

Index