VesselControlSystem

1

Generated by Doxygen 1.8.1

Sun Jun 10 2012 19:40:16

Contents

Chapter 1

Class 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

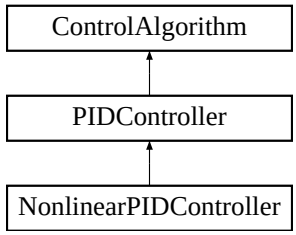
Class Documentation

3.1 ControlAlgorithm Class Reference

Motion controller superclass.

`#include <ControlAlgorithm.h>`

Inheritance diagram for ControlAlgorithm:



Public Member Functions

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW ControlAlgorithm (const string &simObjectName, ISimObjectCreator ∗const creator)

  < *enables the use of eigen types as members variables*
- virtual void OdeFcn (const double T, const double ∗const X, double ∗const XDot, const bool bIsMajorTimeStep)

  Returns object derivatives as a function of time, states and input ports.
- virtual const double ∗const ForcesAndMomentsBODY (const double T, const double ∗const X)=0

  Returns the 6DOF forces and moments calculated by the controller.
- virtual const double ∗const Error (const double T, const double ∗const X)=0

  Returns the error in the position in 6DOF.

Protected Member Functions

- double ShortestError (double eta, double etaRef)

  Calculate shortest angle error in range $-\pi$ - π.
- double RAD2PIPI (double angleIn)

  Converts angle given the range $-\infty$ - ∞ to $-\pi$ - π.
- Eigen::Matrix< double, 3, 3 > RMatrix (const double *const eta)

  Calculates the 3DOF rotation matrix.
- Eigen::Matrix< double, 3, 3 > TMatrix (const double *const eta)

  Calculates the 3DOF transformation matrix.
- Eigen::Matrix< double, 6, 6 > JMatrix (const double *const eta)

  Calculates the 6DOF transformation matrix.

Protected Attributes

- ISignalPort * inputEta

  Input port for the vessel NED position.
- ISignalPort * inputEtaDesired

  Input port for the desired NED position.
- ISignalPort * inputNu

  Input port for the vessel BODY velocity.
- ISignalPort * inputNuDesired

  Input port for the desired BODY velocity.
- ISignalPort * inputNuDotDesired

  Input port for the desired BODY acceleration.
- double tauLimitMin [6]

  Saturation-limits for positive thrust.
- double tauLimitMax [6]

  Saturation-limits for negative thrust.
- double controllableStates [6]

  Array holding a flag for the controllable states.

3.1.1 Detailed Description

Motion controller superclass.

Author

Mats Nk Hval

Superclass for motion control system.es

Revision history:

06.06.2012 MNH: Initial version

User Documentation:

Superclass for motion controllers

Parameters

Parameter	Size	Comment
TauSaturationLimitMax	6	Maximum available force in all 6DOF
TauSaturationLimitMin	6	Minimum available force in all 6DOF
ControllableStates	6	1 or 0, indicating if the state is controllable

Input ports

Tag	Size	Comment
VesselPosition	6	Position and attitude of the vessel.
VesselVelocityBODY	6	Velocity of the vessel in BODY-frame
VesselPositionDesired	6	Desired position vessel
VesselVelocityBODY-Desired	6	Desired velocity of the vessel in BODY-frame
VesselAcceleration-Desired	6	Desired acceleration of the vessel in BODY-frame

Output ports

Tag	Size	Comment
ForcesAndMomentsBO-DY	6	Forces and moments in body frame
Error	6	Position and attitude error

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ControlAlgorithm::ControlAlgorithm (const string & *simObjectName,* ISimObjectCreator ∗const *creator*)

< enables the use of eigen types as members variables

Reads parameters, registers states, input/output ports and shared resources

```
This constructor performs all initial setup for a Control algorithm simobject.
Reading in parameters, setting up communication interface i.e. output ports,
input ports, and states, plus additional 'one time only' resource setup.
```

Parameters

in	*simObject-Name*	The name of the simobject. Used primarily by superclass constructor
in	*creator*	Retrive parameters. Register states, ports and shared resources

3.1.3 Member Function Documentation

3.1.3.1 Eigen::Matrix< double, 6, 6 > ControlAlgorithm::JMatrix (const double *const *eta) [protected]

Calculates the 6DOF transformation matrix.

```
Calculates the rotation matrix in 6DOF using Euler angles.
```

Parameters

in	*eta*	Array containing the 6DOF position.

Returns

Rotation matrix as a 6 Eigen Matrix.

3.1.3.2 double ControlAlgorithm::RAD2PIPI (double *angleIn*) [protected]

Converts angle given the range $-\infty$- ∞ to $-\pi$- π.

```
Converts and angle given the range \form#0- \form#1 to \form#2- \form#3.
```

Parameters

in	*angleIn*	Angle in radians to convert

Returns

angle

3.1.3.3 Eigen::Matrix< double, 3, 3 > ControlAlgorithm::RMatrix (const double *const *eta*) [protected]

Calculates the 3DOF rotation matrix.

```
Calculates the rotation matrix in 3DOF using Euler angles.
```

Parameters

in	*eta*	Array containing the 6DOF position.

Returns

Rotation matrix as a 3 Eigen Matrix

3.1.3.4 double ControlAlgorithm::ShortestError (double *angle,* double *desiredAngle*) [protected]

Calculate shortest angle error in range $-\pi$- π.

```
Method for calculating the shortest rotation error if the angles are defined ir
```

Parameters

in	*angle*	angle in radians
in	*desiredAngle*	Angle in radians

3.1.3.5 Eigen::Matrix< double, 3, 3 > ControlAlgorithm::TMatrix (const double *const *eta*) [protected]

Calculates the 3DOF transformation matrix.

```
Calculates the transformation matrix in 3DOF using Euler angles.
```

Parameters

in	*eta*	Array containing the 6DOF position.

Returns

Transformation matrix as a 3 Eigen Matrix

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

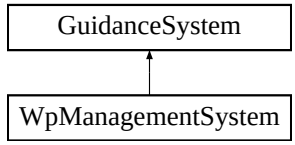
- ControlAlgorithm.h
- ControlAlgorithm.cpp

3.2 GuidanceSystem Class Reference

Superclass of guidance systems for imput to the motion control system Simulation model of a mass point in three degrees of freedom.

`#include <GuidanceSystem.h>`

Inheritance diagram for GuidanceSystem:



Public Member Functions

- GuidanceSystem (const string &simObjectName, ISimObjectCreator ∗const creator)

  reads parameters, registers states, input/output ports and shared resources
- void OdeFcn (const double T, const double ∗const X, double ∗const XDot, const bool bIsMajorTimeStep)

  returns object derivatives as a function of time, states and input ports
- const double ∗const EtaDesired (const double T, const double ∗const X)

  method for returning desired eta (6DOF)
- const double ∗const NuDesired (const double T, const double ∗const X)

  method for returning desired nu (6DOF)
- const double ∗const NuDotDesired (const double T, const double ∗const X)

  method for returning desired nuDot (6DOF)
- const double ∗const EtaReference (const double T, const double ∗const X)

  method for returning the reference points (6DOF)

Public Attributes

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  enables the use of eigen types as member variables

Protected Attributes

- ISignalPort * etaInputPort

  input port. Current position
- ISignalPort * nuInputPort

  input port. Current linear velocity
- double eta [6]

  Position and attitude for the vessel at current time.
- double etaReference [6]

  Input reference position and attitude.
- double startEta [6]

3.2.1 Detailed Description

Superclass of guidance systems for imput to the motion control system Simulation model of a mass point in three degrees of freedom.

//

/

Parameters

Parameter	Default	Comment
Mass	-	The mass of the mass object.
Material	"DiffuseSpecular"	The visualization material of the mass, specified by its name.
Radius	-	Visual radius of sphere
Color	-	Primary color of sphere

Input ports

Tag	Size	Comment
PositionIn	3	The current position of the vessel in NED-frame
VelocityIn	3	The current velocity of the vessel in the BODY frame

Output ports

Tag	Size	Comment
EtaDesired	3	The desired position and attitude in NED
NuDesired	3	The desired velocity in BODY

NuDotDesired	3	The desired acceleration in BODY
EtaReference	3	The reference point in NED

States

Tag	Size	Comment
Eta	3	The desired position.
EtaDot	3	The desired velocity.
EtadDot	3	The desired acceleration.

Author

Mats N{}vik Hval

Revision history:

07.05.2012 MNH: Initial version.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 GuidanceSystem::GuidanceSystem (const string & *simObjectName,* ISimObjectCreator ∗const *creator*)

reads parameters, registers states, input/output ports and shared resources

```
This constructor performs all initial setup for a guidance system simobject.
Reading in parameters, setting up communication interface i.e. output ports,
input ports, and states, plus additional 'one time only' resource setup.
```

Parameters

in	*simObject-Name*	The name of the simobject. Used primarily by superclass constructor
in	*creator*	Retrive parameters. Register states, ports and shared resources

3.2.3 Member Function Documentation

3.2.3.1 const double ∗const GuidanceSystem::EtaDesired (const double *T,* const double ∗const *X*)

method for returning desired eta (6DOF)

```
Returns the desired eta
```

Parameters

in	*T*	dummy
in	*X*	current state

Returns

array of desired eta

3.2.3.2 const double *const GuidanceSystem::EtaReference (const double *T,* const double *const *X*)

method for returning the reference points (6DOF)

```
Returns the desired eta
```

Parameters

in	*T*	dummy
in	*X*	current state

Returns

array of reference eta

3.2.3.3 const double *const GuidanceSystem::NuDesired (const double *T,* const double *const *X*)

method for returning desired nu (6DOF)

```
Returns the desired eta
```

Parameters

in	*T*	dummy
in	*X*	current state

Returns

array of desired nu

3.2.3.4 const double *const GuidanceSystem::NuDotDesired (const double *T,* const double *const *X*)

method for returning desired nuDot (6DOF)

```
Returns the desired eta
```

Parameters

in	*T*	dummy
in	*X*	current state

Returns

array of desired nuDot

3.2.3.5 void GuidanceSystem::OdeFcn (const double *T,* const double *const *X,* double *const *XDot,* const bool *bIsMajorTimeStep*)

returns object derivatives as a function of time, states and input ports

implements a third order filter based reference model.

Reimplemented in WpManagementSystem.

3.2.4 Member Data Documentation

3.2.4.1 double GuidanceSystem::startEta[6] [protected]

array holding eta at the time a new setpoint is set

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

- GuidanceSystem.h
- GuidanceSystem.cpp

3.3 NetNodeStruct Struct Reference

A struct holding information on the different nodes in the netstructure.

```
#include <WpManagementSystem.h>
```

Public Attributes

- Waypoint ∗ WP
  pointer to the moovable waypoint
- NetNodeStruct ∗ Left
  pointer to the netnode to the left
- NetNodeStruct ∗ Right
  pointer to the netnode to the right
- bool visited
  flag to tell if the node already is visited
- bool sorted
  flag to tell if the node has been sorted

3.3.1 Detailed Description

A struct holding information on the different nodes in the netstructure.

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

- WpManagementSystem.h

3.4 Nonlinear Class Reference

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

- NonlinearPassiveObserver.h

3.5 NonlinearPassiveObserver Class Reference

Nonlinear Passive Observer as described in Fossen 2011.

`#include <NonlinearPassiveObserver.h>`

Public Member Functions

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW NonlinearPassiveObserver (const string &simObjectName, ISimObjectCreator ∗const creator)
  reads parameters, registers states, input/output ports and shared resources
- void OdeFcn (const double T, const double ∗const X, double ∗const XDot, const bool bIsMajorTimeStep)
  returns object derivatives as a function of time, states and input ports

- const double *const AccelerationBODY (const double T, const double *const X)
- const double *const VelocityBODY (const double T, const double *const X)
- const double *const VelocityNED (const double T, const double *const X)
- const double *const PositionNED (const double T, const double *const X)
- const double *const PositionNEDwithWaves (const double T, const double *const X)
- const double *const BiasEstimate (const double T, const double *const X)
- const double *const YError (const double T, const double *const X)

Protected Attributes

- ISignalPort * EtaInputPort

  input port. Current position and attitude
- ISignalPort * ThrusterInputPort

  input port. Thruster forces and moments
- ISignalPort * MooringForcesInputPort

  input port. Forces and moments from optional mooring lines
- ISignalPort * StateMeasurementEta

  input port
- bool waveFiltering

  Is the wave filter active?
- double stateEtaEstimate [6]

  Estimate of the position and attitude.
- double stateBiasEstimate [6]

  Estimate of the slowly varying bias.
- double stateWaveEstimate [12]

  Estimate of the waves.
- double etaMeasurement [6]

  Measurements of the vessel position and attitude.
- double etaError [6]

  Estimation error. EtaMeasurement - EtaEstimate.
- int indexWaveEstimate

  index of the wave estimate in the state vector
- int indexBiasEstimate

  index of the bias estimate in the state vector
- int indexNuEstimate

  index of nuHat in the state vector
- int indexEtaEstimate

  index of etaHat in the vector
- double accelerationBODY [6]

  return array for the BODY acceleration
- double velocityBODY [6]

return array for the BODY velocity

- double velocityNED [6]

  return array for the NED velocity
- double positionNED [6]

  return array for the NED position
- double positionNEDwithWaves [6]

  return array for the NED position with waves
- double yTilde [6]

  return array for the error between measured and estimated position
- double waveModel_omega [6]

  wave model parameters, omea
- double waveModel_delta [6]

  wave model parameters, delta
- double bias_timeConstants [6]

  bias model parameters, timeconstants
- double linearDampingDiagonal [6]

  diagonal elements of the linear damping
- double nonlinearDampingDiagonal [6]

  diagonal elements of the quadratic damping
- double massMatrixDiagonal [6]

  diagonal elements of the combined rigid body and added mass matrix
- double aK1 [12]

  Diagonal elements of K1 tuning matrix.
- double aK2 [6]

  Diagonal elements of K2 tuning matrix.
- double aK3 [6]

  Diagonal elements of K3 tuning matrix.
- double aK4 [6]

  Diagonal elements of K4 tuning matrix.
- Eigen::Matrix< double, 12, 6 > K1

  K1 tuning matrix of the wave model.
- Eigen::Matrix< double, 12, 12 > Aw

  Aw model matrix of the wave model.
- Eigen::Matrix< double, 6, 12 > Cw

  CW wave term measurement matrix of the wave model.
- Eigen::Matrix< double, 6, 6 > waveOmega

  wave model omega
- Eigen::Matrix< double, 6, 6 > waveDelta

  Wave model delta.
- double waveNumber

  wavenumber of the wave with the largest impact

- Eigen::Matrix< double, 6, 6 > K3

  K3 tuning matrix for the bias model.
- Eigen::Matrix< double, 6, 6 > Tinv

  Iverse bias time constant matrix.
- double bias [6]

  return array for the bias estimate
- double biasExponentialTermMax [6]

  maxvalue of the bias exponential term.
- double biasExponentialTermTimeConstants [6]

  time constant of the bias exponential term
- Eigen::Matrix< double, 6, 6 > Dlinear

  linear damping matrix
- Eigen::Matrix< double, 6, 6 > Dnonlinear

  quadratic damping matrix
- Eigen::Matrix< double, 6, 6 > D

  sum of linear and quadratic damping
- Eigen::Matrix< double, 6, 6 > J

  6DOF transformation matrix
- Eigen::Matrix< double, 6, 6 > Jinv

  inverse 6DOF transformation matrix
- Eigen::Matrix< double, 6, 6 > K4

  K4 tuning matrix.
- Eigen::Matrix< double, 6, 6 > Minv

  inverse mass matrix
- Eigen::Matrix< double, 6, 6 > K2

  K2 tuning matrix.
- Eigen::Matrix< double, 6, 1 > Yhat

  measurement estimate vector
- Eigen::Matrix< double, 6, 1 > Y

  measurement vector
- Eigen::Matrix< double, 6, 1 > Ytilde

  Estimate error vector.
- Eigen::Matrix< double, 6, 1 > VbiasEstimate

  bias estimate vector
- Eigen::Matrix< double, 6, 1 > VbiasEstimateDot

  estimateDot vector
- Eigen::Matrix< double, 6, 1 > VetaEstimate

  etaEstimate vector
- Eigen::Matrix< double, 6, 1 > VetaEstimateDot

  eta dot estimate vectors
- Eigen::Matrix< double, 6, 1 > VnuEstimate

nu estimate vector

- Eigen::Matrix< double, 6, 1 > VnuEstimateDot

  nudot estimate vector
- Eigen::Matrix< double, 12, 1 > VwaveEstimate

  wave estimate vector
- Eigen::Matrix< double, 12, 1 > VwaveEstimateDot

  waveestimateDot vector
- Eigen::Matrix< double, 6, 1 > ThrusterForcesAndMoments

  thruster forces and moments vector
- Eigen::Matrix< double, 6, 1 > MooringForcesAndMoments

  mooring loads vector

3.5.1 Detailed Description

Nonlinear Passive Observer as described in Fossen 2011.

Author

Mats N{}vik Hval

3.5.2 Constructor & Destructor Documentation

3.5.2.1 NonlinearPassiveObserver::NonlinearPassiveObserver (const string & *simObjectName,* ISimObjectCreator *const *creator*)

reads parameters, registers states, input/output ports and shared resources

Author

Mats N{}vik Hval

```
Nonlinear Passive Observer for a 6DOF vessel
```

Revision history:

06.06.2012 MNH: Initial version. This constructor performs all initial setup for a nonlinear passive observer simobject. Reading in parameters, setting up communication interface i.e. output ports, input ports, and states, plus additional 'one time only' resource setup.

Parameters

in	*simObject-Name*	The name of the simobject. Used primarily by superclass constructor
in	*creator*	Retrive parameters. Register states, ports and shared resources

3.5.3 Member Function Documentation

3.5.3.1 const double *const NonlinearPassiveObserver::AccelerationBODY (const double *T,* const double *const *X*)

```
Returns the acceleration
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array containing 6DOF accelerations in BODY frame

3.5.3.2 const double *const NonlinearPassiveObserver::BiasEstimate (const double *T,* const double *const *X*)

```
Returns the bias estimate
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array containing the bias estimate

3.5.3.3 const double *const NonlinearPassiveObserver::PositionNED (const double *T,* const double *const *X*)

```
Returns the filtered position in NED-frame
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array containing filtered 6OF position in NED

3.5.3.4 const double *const NonlinearPassiveObserver::PositionNEDwithWaves (const double *T,* const double *const *X*)

```
Sorts a vector of netnodes on ascending north position
```

Parameters

in	*T*	current simulation time
in	*X*	current simulation state

Returns

array containing 6DOF position and attitude with wave disturbance

3.5.3.5 const double *const NonlinearPassiveObserver::VelocityBODY (const double *T,* const double *const *X*)

```
Returns the velocity in BODY frame
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array containing 6DOF velocities in BODY frame

3.5.3.6 const double *const NonlinearPassiveObserver::VelocityNED (const double *T,* const double *const *X*)

```
Returns the velocity in NED-frame
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array containing 6DOF velocity in NED

3.5.3.7 const double *const NonlinearPassiveObserver::YError (const double *T*, const double *const *X*)

Returns the error between measured and estimated position. y-yHat.

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array containing error between measured and estimated position

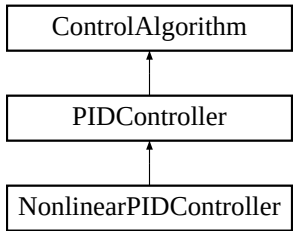
The documentation for this class was generated from the following files:

- NonlinearPassiveObserver.h
- NonlinearPassiveObserver.cpp

3.6 NonlinearPIDController Class Reference

`#include <NonlinearPIDController.h>`

Inheritance diagram for NonlinearPIDController:



Public Member Functions

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW NonlinearPIDController (const string &simObjectName, ISimObjectCreator *const creator)

  reads parameters, registers states, input/output ports and shared resources

- const double *const ForcesAndMomentsBODY (const double T, const double *const X)

  Returns calculated forces and moments.

- void OdeFcn (const double T, const double *const X, double *const XDot, const bool bIsMajorTimeStep)

  returns object derivatives as a function of time, states and input ports

Additional Inherited Members

3.6.1 Detailed Description

User Documentation:

Parameters

Parameter	Size	Comment
SaturationFactor_Tau-Integral	6	Factor indicating how large percentage the integral effect should be given of the available force
RigidBodyandAdded-MassMatrixDiagonal	6	Rigid body and added mass matrix diagonal
LinearDampingMatrix-Diagonal	6	Linear damping matrix diagonal
NonlinearDamping-MatrixDiagonal	6	quadratic damping diagonal
IntegralLimits	6	limits for when the integrator starts to integrate
AntiWindupConstants	6	Constants for the anti windup effect
TauSaturationLimitMin	6	Minimum available force in all 6DOF
TauSaturationLimitMax	6	Maximum available force in all 6DOF

3.6.2 Member Function Documentation

3.6.2.1 const double *const NonlinearPIDController::ForcesAndMomentsBODY (const double *T*, const double *const *X*) [virtual]

Returns calculated forces and moments.

```
Returns the forces and moments desired from the controller
```

Only the yaw-torque is implemented.

Parameters

in	*T*	current simulation time
in	*X*	current simulation state

Returns

desired forces and momemnts

Reimplemented from PIDController.

3.6.2.2 void NonlinearPIDController::OdeFcn (const double *T*, const double *const *X*, double *const *XDot*, const bool *bIsMajorTimeStep*) [virtual]

returns object derivatives as a function of time, states and input ports

```
Calculates the error in position and velocity, and assignes the position error to the s
```

The error is defined as value - desired_value.

Parameters

in	*T*	current simulation time
in	*X*	current simulation state
in	*XDot*	pointer to the integrand
in	*bIsMajor-TimeStep*	is this a major time step

Reimplemented from PIDController.

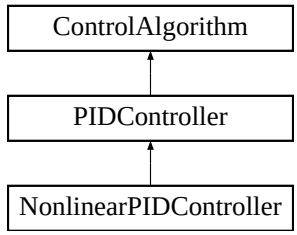
The documentation for this class was generated from the following files:

- NonlinearPIDController.h
- NonlinearPIDController.cpp

3.7 PIDController Class Reference

`#include <PIDController.h>`

Inheritance diagram for PIDController:



Public Member Functions

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW PIDController (const string &simObjectName, ISimObjectCreator *const creator)

  Reads parameters, registers states, input/output ports and shared resources.
- const double *const ForcesAndMomentsBODY (const double T, const double *const X)

  Implements the forces and moments calculation.
- void OdeFcn (const double T, const double *const X, double *const XDot, const bool bIsMajorTimeStep)

  Returns object derivatives as a function of time, states and input ports.
- const double *const Error (const double T, const double *const X)

  Implements return of the error in position.
- const double *const ErrorIntegral (const double T, const double *const X)

  Returns the integral of the error.
- const double *const TauProp (const double T, const double *const X)

  Returns the proportional term of the controller.
- const double *const TauInt (const double T, const double *const X)

  Returns the integral term of the controller.
- const double *const TauDamp (const double T, const double *const X)

  Returns the derivative term of the controller.

Protected Member Functions

- virtual void CalculateTauP ()

  Calculate the proportional term.
- virtual void CalculateTauI ()

  Calculate the integral term.
- virtual void CalculateTauD ()

  Calculate the derivative term.

Protected Attributes

- Eigen::Matrix< double, 6, 6 > Kp

  Diagonal matrix holding the porportional controller gain.
- Eigen::Matrix< double, 6, 6 > Ki

  Diagonal matrix holding the porportional controller gain.
- Eigen::Matrix< double, 6, 6 > Kd

  Diagonal matrix holding the porportional controller gain.
- Eigen::Matrix< double, 6, 1 > error

  Vector of position error, for calculations.
- Eigen::Matrix< double, 6, 1 > errorDot

  Vector of error in velocity, for calculations.
- Eigen::Matrix< double, 6, 1 > errorIntegral

  Vector of error-integral, for calculations.
- Eigen::Matrix< double, 6, 1 > force

  Resulting forces and moments, for calculations.
- int indexErrorIntegral

  Index for the error-integral-state in the state vector.
- double forcesAndMomentsBODY [6]

  Return array for body forces and moments.
- double a_error [6]

  Return array for the error.
- double a_errorIntegral [6]

  Return array for the error integral.
- double integralLimits [6]

  Array holding integral limits.
- double a_tauP [6]

  Return array for the proportional term forces and moments.
- double a_tauI [6]

  Return array for the integral term forces and moments.
- double a_tauD [6]

  Return array for the derivative term forces and moments.

3.7.1 Detailed Description

User Documentation:

Simple PID controller

Parameters

Parameter	Size	Comment
ProportionalGainVector	6	Coefficients for the proporional gain
IntegralGainVector	6	Coefficients for the integral gain
DerivativeGainVector	6	Coefficients for the derivative gain

Output ports

Tag	Size	Comment
ErrorIntegral	6	Integral of the error
TauP	6	Proporional term of the controller
TauI	6	Integral term of the controller
TauD	6	Derivative term of the controller

3.7.2 Member Function Documentation

3.7.2.1 void PIDController::CalculateTauD ( ) [protected], [virtual]

Calculate the derivative term.

Calculates the forces in the NED frame proporional with the derivative of the error

3.7.2.2 void PIDController::CalculateTauI ( ) [protected], [virtual]

Calculate the integral term.

Calculates the forces in the NED frame proportional with the integral of the error

3.7.2.3 void PIDController::CalculateTauP ( ) [protected], [virtual]

Calculate the proportional term.

Calculates the forces in the NED frame proportional with the error

3.7.2.4 const double *const PIDController::Error (const double *T*, const double *const *X*) [virtual]

Implements return of the error in position.

```
Returns the error-array
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array of errors in position and attitude

Implements ControlAlgorithm.

3.7.2.5 const double *const PIDController::ErrorIntegral (const double *T*, const double *const *X*)

Returns the integral of the error.

```
Returns the errorintegral-vector
```

Parameters

in	*T*	dummy
in	*X*	current simulation state

Returns

integral of the error.

3.7.2.6 const double *const PIDController::ForcesAndMomentsBODY (const double *T*, const double *const *X*) [virtual]

Implements the forces and moments calculation.

```
Returns the forces and moments desired from the controller
```

Parameters

in	*T*	current simulation time
in	*X*	current simulation state

return desired forces and moments in 6DOF

Implements ControlAlgorithm.

Reimplemented in NonlinearPIDController.

3.7.2.7 void PIDController::OdeFcn (const double *T*, const double *const *X*, double *const *XDot*, const bool *bIsMajorTimeStep*) [virtual]

Returns object derivatives as a function of time, states and input ports.

```
Calculates the error in position and velocity, and assignes the position error to the s
```

The error is defined as value - desired_value.

Parameters

in	*T*	current simulation time
in	*X*	current simulation state
in,out	*XDot*	derivative of the state
in	*bIsMajor-TimeStep*	major time step flag

Reimplemented from ControlAlgorithm.

Reimplemented in NonlinearPIDController.

3.7.2.8 const double *const PIDController::TauDamp (const double *T*, const double *const *X*)

Returns the derivative term of the controller.

```
Returns the proportional term
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array of the derivative term

3.7.2.9 const double *const PIDController::TauInt (const double *T*, const double *const *X*)

Returns the integral term of the controller.

```
Returns the proportional term
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array of the integral term

3.7.2.10 const double *const PIDController::TauProp (const double *T*, const double *const *X*)

Returns the proportional term of the controller.

```
Returns the proportional term
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

array of the proportional term

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

- PIDController.h
- PIDController.cpp

3.8 SampledSignalWithNoise Class Reference

```
#include <SampledSignalWithNoise.h>
```

Public Member Functions

- SampledSignalWithNoise (const string &simObjectName, ISimObjectCreator *const creator)

  reads parameters, registers states, input/output ports and shared resources
- const double *const Signal (const double T, const double *const X)

  returns the distorted signal
- void OdeFcn (const double T, const double *const X, double *const XDot, const bool bIsMajorTimeStep)

  returns object derivatives as a function of time, states and input ports

3.8.1 Detailed Description

```
Sampled the signals at a given sampling rate, and outputs as a ZOH sign
```

Author

Mats N{}vik Hval

Revision history:

29.04.2012 MNH: Initial version.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 SampledSignalWithNoise::SampledSignalWithNoise (const string & *simObjectName,* ISimObjectCreator ∗const *creator*)

reads parameters, registers states, input/output ports and shared resources

```
This constructor performs all initial setup for the referencegenerator simobjec
Reading in parameters, setting up communication interface i.e. output ports,
input ports, and states, plus additional 'one time only' resource setup.
```

Parameters

in	*simObject-Name*	The name of the simobject. Used primarily by superclass constructor
in	*creator*	Retrieve parameters. Register states, ports and shared resources

3.8.3 Member Function Documentation

3.8.3.1 const double ∗const SampledSignalWithNoise::Signal (const double *T,* const double ∗const *X*)

returns the distorted signal

```
Returns the sampled signal with noise
```

///

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

return the array of distorted signals

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

- SampledSignalWithNoise.h
- SampledSignalWithNoise.cpp

3.9 ThrusterAllocation Class Reference

Public Member Functions

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW ThrusterAllocation (const string &simObjectName, ISimObjectCreator ∗const creator)

  < enabling use of Eigen types as members
- const double ∗const RPM (const double T, const double ∗const X)

  returns array of commanded RPMs
- const double ∗const ThrustCommanded (const double T, const double ∗const X)

  returns array of corresponding tau
- void OdeFcn (const double T, const double ∗const X, double ∗const XDot, const bool bIsMajorTimeStep)

  returns object derivatives as a function of time, states and input ports

3.9.1 Constructor & Destructor Documentation

3.9.1.1 ThrusterAllocation::ThrusterAllocation (const string & *simObjectName,* ISimObjectCreator ∗const *creator*)

< enabling use of Eigen types as members

reads parameters, registers states, input/output ports and shared resources

3.9.2 Member Function Documentation

3.9.2.1 void ThrusterAllocation::OdeFcn (const double *T,* const double ∗const *X,* double ∗const *XDot,* const bool *bIsMajorTimeStep*)

returns object derivatives as a function of time, states and input ports

```
Calculates the error in position and velocity, and assignes the position error to the s
```

The error is defined as value - desired_value.

Parameters

in	*T*	current simulation time
in	*X*	current simulation state
in	*XDot*	pointer to the integrand
in	*bIsMajor-TimeStep*	major time step flag

3.9.2.2 const double *const ThrusterAllocation::RPM (const double *T,* const double *const *X*)

returns array of commanded RPMs

```
Returnes the calculated RPMs for the thrusters
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

RPM

3.9.2.3 const double *const ThrusterAllocation::ThrustCommanded (const double *T,* const double *const *X*)

returns array of corresponding tau

```
Calculates the thrust from a second order rpm-to-thrust equation based on the rpm comma
        thrust = coeff1 * rpm^2 + coeff2 * rpm
```

Parameters

in	*T*	dummy
in	*X*	dummy

Returns

tau commanded

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

- ThrusterAllocation.h
- ThrusterAllocation.cpp

3.10 ThrusterStruct Struct Reference

Public Attributes

- double positionAndAttitude [5]

  Position and attitude of the thruster with regards to the vehicles center of origin.
- int RPMlimits [2]

  minimum and maximum rotations per minute per thruster
- double RPMthrustCoeffs [2]

  coefficients describing a second order curve of the relation between RPM and thrust
- double rpm

  calculated rpm
- double thrust

  calculated thrust

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

- ThrusterAllocation.h

3.11 Waypoint Class Reference

Simple waypoint object.

`#include <Waypoint.h>`

Public Member Functions

- Waypoint ()
- Waypoint (double ∗const NPosition, double ∗const EPosition, double ∗const DPosition, double radiusOfAcceptence)
- Waypoint (double NPosition, double EPosition, double DPosition, double radiusOfAcceptance)
- void setAttitude (double attitude[3])
- void setRadiusOfAcceptance (double radius)
- void setPosition (double ∗const n, double ∗const e, double ∗const d)
- void setPosition (double n, double e, double d)
- double ∗ Position ()

  Returns the position of the waypoint.
- double ∗ Attitude ()

  Returns the desired heading when passing the waypoint.
- double ∗ Velocity ()

  Returns the desired velocity when passing the waypoint.

- double Radius ()

  Returns the radius of acceptence. From within which radius of the current waypoint can the waypointchange occur.

3.11.1 Detailed Description

Simple waypoint object.

Author

Mats N{}vik Hval

Revision history:

28.06.2011 MNH: Initial version. 06.06.2012 MNH: v2

3.11.2 Constructor & Destructor Documentation

3.11.2.1 Waypoint::Waypoint ( )

This constructor registers the position and radius of acceptence for a waypoint to be zero

3.11.2.2 Waypoint::Waypoint (double ∗const *nPosition,* double ∗const *ePosition,* double ∗const *dPosition,* double *RadiusOfAcceptance*)

```
This constructor registers the position and radius of acceptence for a waypoint
```

Parameters

in	*nPosition*	The N-position of the waypoint
in	*ePosition*	The E-position of the waypoint
in	*dPosition*	The D-position of the waypoint
in	*RadiusOf-Acceptance*	The radius of acceptance of the waypoint

3.11.2.3 Waypoint::Waypoint (double *n,* double *e,* double *d,* double *radius*)

```
This constructor registers the position and radius of acceptence for a waypoint
```

Parameters

in	*n*	The N-position of the waypoint
in	*e*	The E-position of the waypoint

in	*d*	The D-position of the waypoint
in	*radius*	The radius of acceptance of the waypoint

3.11.3 Member Function Documentation

3.11.3.1 double * Waypoint::Attitude ( )

Returns the desired heading when passing the waypoint.

```
Returns the desired attitude when passing the waypoint.
```

Returns

desired attitude at the waypoint

< Returns the desired heading when passing the waypoint

3.11.3.2 double * Waypoint::Position ( )

Returns the position of the waypoint.

```
Output port. Returns current position.
```

Returns

position of the waypoint

< Returns a pointer to the position-array of the waypoint

3.11.3.3 double Waypoint::Radius ( )

Returns the radius of acceptence. From within which radius of the current waypoint can the waypointchange occur.

Returns the radius of acceptence

Returns

waypoint radius of acceptence

3.11.3.4 void Waypoint::setAttitude (double *attitude[3]*)

Set the desired attitude at the waypoint

Parameters

in	*attitude*	desired attitude at the way point

3.11.3.5 void Waypoint::setPosition (double *const n, double *const e, double *const d)

Sets the position of the waypoint

```
\param [in]     n       north position
\param [in]     e       east position
\param [in]     d       down position
```

3.11.3.6 void Waypoint::setPosition (double *n, double *e, double *d)

Sets the postion of the waypoint

```
\param [in]     n       north position
\param [in]     e       east position
\param [in]     d       down position
```

3.11.3.7 void Waypoint::setRadiusOfAcceptance (double *radius*)

Sets the radius of acceptance for the way point

Parameters

in	*radius*	radius of acceptance for the way point

3.11.3.8 double * Waypoint::Velocity ( )

Returns the desired velocity when passing the waypoint.

```
Output port. Returns the desired velocity when passing way point
```

Returns

Desired velocity when passing wapoint

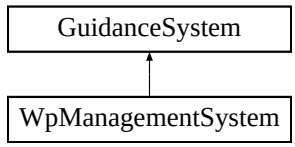
< Returns the desired speed when passing the waypoint

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

- Waypoint.h
- Waypoint.cpp

3.12 WpManagementSystem Class Reference

Inheritance diagram for WpManagementSystem:



Public Member Functions

- WpManagementSystem (const string &simObjectName, ISimObjectCreator ∗const creator)

  reads parameters, registers states, input/output ports and shared resources
- void OdeFcn (const double T, const double ∗const X, double ∗const XDot, const bool bIsMajorTimeStep)

  returns object derivatives as a function of time, states and input ports
- void FinalSetup (const double T, const double ∗const X, ISimObjectCreator ∗const creator)

  sets up the nodes if cage traversing is the chosen method

Protected Member Functions

- std::vector< NetNodeStruct ∗ > quicksort (std::vector< NetNodeStruct ∗ > inputVector)

  method for sorting the nodes at each level.
- void findNeighboringNodes (std::vector< NetNodeStruct ∗ > nodes)

  method for find the neighboring nodes of each node.

Protected Attributes

- std::string m_NetCageName

  name of the netstructure
- double radiusOfAcceptance

radius of acceptance of the node

- std::vector< Waypoint * > wpVector

  vector of the predefined waypoints
- std::vector< std::vector < NetNodeStruct * > > netcage

  vector holding the nodes in the net structure
- DeepSeaGravityWaves * m_Environment

  pointer to the current environment
- bool predefinedWP

  flag for predefined wps
- bool netcageTraversing

  flag for netcage traversing
- bool traverseUpNdown

  flag for traversing route
- bool traverseCircles

  flag for traversing route
- Waypoint * currentWaypoint

  current desired wp
- int waypointIndex

  index of the wp
- bool atWP

  flag to tell if the WP is reached
- NetNodeStruct * currentNetNode

  current desired netnode
- double cageDiameter

  diameter of the cage
- int level

  telling which level in the cage which is interesting
- double angleTol

  tolerance for the heading error

Additional Inherited Members

3.12.1 Constructor & Destructor Documentation

3.12.1.1 WpManagementSystem::WpManagementSystem (const string & *simObjectName,* ISimObjectCreator *const *creator*)

reads parameters, registers states, input/output ports and shared resources

```
This constructor performs all initial setup for a reference generator simobject
Reading in parameters, setting up communication interface i.e. output ports,
input ports, and states, plus additional 'one time only' resource setup.
```

Parameters

in	*simObject-Name*	The name of the simobject. Used primarily by superclass constructor
in	*creator*	Retrive parameters. Register states, ports and shared resources

3.12.2 Member Function Documentation

3.12.2.1 void WpManagementSystem::FinalSetup (const double *T,* const double *const *X,* ISimObjectCreator *const *creator*)

sets up the nodes if cage traversing is the chosen method

```
This method registers the netcage nodes
```

Parameters

in	*T*	Current simulation time
in	*X*	Current simulation states
in	*creator*	Retrive parameters. Register states, ports and shared resources

3.12.2.2 void WpManagementSystem::findNeighboringNodes (std::vector< NetNodeStruct * > *nodes*) [protected]

method for find the neighboring nodes of each node.

```
finds and sets the right and left neighboring nodes of each node in the vector
```

Parameters

in	*nodes*	vector of the nodes at one depth level in the cage.

Returns

vector of sorted nodes

3.12.2.3 void WpManagementSystem::OdeFcn (const double *T,* const double *const *X,* double *const *XDot,* const bool *bIsMajorTimeStep*)

returns object derivatives as a function of time, states and input ports

```
Calculates the desired position and changes the setpoint when inside the circle of acce
```

The error is defined as value - desired_value.

Parameters

in	*T*	current simulation time
in	*X*	current simulation state
in	*XDot*	pointer to the integrand
in	*bIsMajor-TimeStep*	is this a major time step

```
PredefinedWP
```

```
NetCageTraversing
```

Reimplemented from GuidanceSystem.

3.12.2.4 std::vector< NetNodeStruct * > WpManagementSystem::quicksort (std::vector< NetNodeStruct * > *inputVector*) [protected]

method for sorting the nodes at each level.

```
Sorts a vector of netnodes on ascending north position
```

Parameters

in	*inputVector*	vector of the nodes at one depth level in the cage.

Returns

vector of sorted nodes

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

- WpManagementSystem.h
- WpManagementSystem.cpp

Index