UAV physic engine

Generated by Doxygen 1.9.1

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 controllers Namespace Reference

Classes

- class BangBang
- class DoubleSetpoint
- class PID
- class PID_Discrete
- class ZTransformStatic
- class ZTransform

5.2 def Namespace Reference

Simulation constants.

Variables

- const double GRAVITY_CONST = 9.81

  Gravity constant on Earth in m/s2.
- const double FRICTION_EPS = 0.001

  minimal friction that is calculated (numerical float eps)
- const double GENTLY_PUSH = 0.15

  artificial force coefficient. Protect again diving objects in horizontal wall
- const double DOUBLE_EPS = 1e-5

  near zero floating point eps
- const double MIXING_FUNCTION = 0.1

  mixing window used in blending normal coefficients with standard ones, when stall angle was exceeded
- const int validityOfForce = 5

  how many times outer force should be used

5.2.1 Detailed Description

Simulation constants.

5.2.2 Variable Documentation

5.2.2.1 DOUBLE_EPS

```
const double def::DOUBLE_EPS = 1e-5
```

near zero floating point eps

5.2.2.2 FRICTION_EPS

```
const double def::FRICTION_EPS = 0.001
```

minimal friction that is calculated (numerical float eps)

5.2.2.3 GENTLY_PUSH

```
const double def::GENTLY_PUSH = 0.15
```

artificial force coefficient. Protect again diving objects in horizontal wall

5.2.2.4 GRAVITY_CONST

```
const double def::GRAVITY_CONST = 9.81
```

Gravity constant on Earth in m/s2.

5.2.2.5 MIXING_FUNCTION

```
const double def::MIXING_FUNCTION = 0.1
```

mixing window used in blending normal coefficients with standard ones, when stall angle was exceeded

5.2.2.6 validityOfForce

```
const int def::validityOfForce = 5
```

how many times outer force should be used

5.3 zmq_recv Namespace Reference

Variables

- context = zmq.Context()
- socket = context.socket(zmq.SUB)
- string topicfilter = "pos"
- s = socket.recv_string()

5.3.1 Variable Documentation

5.3.1.1 context

```
zmq_recv.context = zmq.Context()
```

5.3.1.2 s

```
zmq_recv.s = socket.recv_string()
```

5.3.1.3 socket

```
zmq_recv.socket = context.socket(zmq.SUB)
```

5.3.1.4 topicfilter

```
string zmq_recv.topicfilter = "pos"
```

5.4 zmq_recv_last Namespace Reference

Variables

- context = zmq.Context()
- socket = context.socket(zmq.SUB)
- string topicfilter = ""
- s = socket.recv_string()

5.4.1 Variable Documentation

5.4.1.1 context

```
zmq_recv_last.context = zmq.Context()
```

5.4.1.2 s

```
zmq_recv_last.s = socket.recv_string()
```

5.4.1.3 socket

```
zmq_recv_last.socket = context.socket(zmq.SUB)
```

5.4.1.4 topicfilter

```
string zmq_recv_last.topicfilter = ""
```

5.5 zmq_send Namespace Reference

Variables

- context = zmq.Context()
- socket = context.socket(zmq.PUB)
- int counter = 0

5.5.1 Variable Documentation

5.5.1.1 context

```
zmq_send.context = zmq.Context()
```

5.5.1.2 counter

```
int zmq_send.counter = 0
```

5.5.1.3 socket

```
zmq_send.socket = context.socket(zmq.PUB)
```

5.6 zmq_send_tcp Namespace Reference

Variables

- context = zmq.Context()
- socket = context.socket(zmq.PUB)
- float angle = 0.0

5.6.1 Variable Documentation

5.6.1.1 angle

```
float zmq_send_tcp.angle = 0.0
```

5.6.1.2 context

```
zmq_send_tcp.context = zmq.Context()
```

5.6.1.3 socket

```
zmq_send_tcp.socket = context.socket(zmq.PUB)
```

Chapter 6

Class Documentation

6.1 AeroCoefficients Struct Reference

Aerodynamic coefficient.

`#include <aero_coefficients.hpp>`

Public Attributes

- double S
- double d
- double eAR
- Eigen::Vector< double, 6 > C0
- Eigen::Matrix< double, 6, 3 > Cpqr
- Eigen::Matrix< double, 6, 4 > Cab
- double stallLimit

6.1.1 Detailed Description

Aerodynamic coefficient.

6.1.2 Member Data Documentation

6.1.2.1 C0

`Eigen::Vector<double,6> AeroCoefficients::C0`

6.1.2.2 Cab

```
Eigen::Matrix<double,6,4> AeroCoefficients::Cab
```

6.1.2.3 Cpqr

```
Eigen::Matrix<double,6,3> AeroCoefficients::Cpqr
```

6.1.2.4 d

```
double AeroCoefficients::d
```

6.1.2.5 eAR

```
double AeroCoefficients::eAR
```

6.1.2.6 S

```
double AeroCoefficients::S
```

6.1.2.7 stallLimit

```
double AeroCoefficients::stallLimit
```

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

- lib/UAV_common/src/components/aero_coefficients.hpp

6.2 AHRSParams Struct Reference

AHRS parameters.

```
#include <navi.hpp>
```

Public Attributes

- std::string type
- double alpha
- double Q
- double R

6.2.1 Detailed Description

AHRS parameters.

6.2.2 Member Data Documentation

6.2.2.1 alpha

```
double AHRSParams::alpha
```

6.2.2.2 Q

```
double AHRSParams::Q
```

6.2.2.3 R

```
double AHRSParams::R
```

6.2.2.4 type

```
std::string AHRSParams::type
```

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

- lib/UAV_common/src/components/navi.hpp

6.3 Aircraft Class Reference

central class in simulation

`#include <aircraft.hpp>`

Collaboration diagram for Aircraft:

Public Member Functions

- Aircraft ()

  Default constructor.
- virtual ~Aircraft ()

  Virtual deconstructor in case of future use in devired class.
- void update ()

  Simulation step.
- void sendState (zmq::socket_t *socket)

  Sends simulation state via publisher socket.
- bool startJet (int index)

  Starts jet engine.
- void trim ()

  Restore trim values of surface angles.
- bool setSurface (Eigen::VectorXd angles)

  Set surface deflation.
- bool setHinge (char type, int index, int hinge_index, double value)

  Set angle of specified hinge in specified drive.
- void calcImpulseForce (double COR, double mi_static, double mi_dynamic, Eigen::Vector3d collisionPoint, Eigen::Vector3d surfaceNormal)

  Calculate impact and result of collision with with solid surface. Results are applied to UAV state.
- std::tuple< int, Eigen::Vector3d > dropCargo (int index)

  Release cargo of specified index.
- std::tuple< int, Eigen::Vector3d > shootAmmo (int index)

  Shoot ammo of specified index.

Public Attributes

- UAVstate state

Protected Member Functions

- void reduceMass (double delta_m, Eigen::Vector3d r)

  Reduces mass of aircraft of given value. Mass matrix is reduced proportionally - moments of inertia is scaled as well.
- Eigen::Vector3d calcMomentumConservanceConservation (double m, Eigen::Vector3d speed, Eigen::↩ Vector3d r)

  Calculateds result of releasing/launching object from aircraft.
- virtual Eigen::VectorXd RHS (double, Eigen::VectorXd)

  Right hand side of main differential equation.

Protected Attributes

- Matrix< double, 6, 6 > massMatrix
- Matrix< double, 6, 6 > invMassMatrix
- std::mutex mtx
- int noOfRotors
- std::unique_ptr< Rotor[ ]> rotors
- int noOfJets
- std::unique_ptr< Jet[ ]> jets
- ControlSurfaces surfaces
- AeroCoefficients aero
- int noOfAmmo
- std::unique_ptr< Ammo[ ]> ammo
- int noOfCargo
- std::unique_ptr< Cargo[ ]> cargo
- std::unique_ptr< ODE > ode

6.3.1 Detailed Description

central class in simulation

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Aircraft()

```
Aircraft::Aircraft ( )
```

Default constructor.

6.3.2.2 ~Aircraft()

```
virtual Aircraft::~Aircraft ( )  [inline], [virtual]
```

Virtual deconstructor in case of future use in devired class.

6.3.3 Member Function Documentation

6.3.3.1 calcImpulseForce()

```
void Aircraft::calcImpulseForce (
            double COR,
            double mi_static,
            double mi_dynamic,
            Eigen::Vector3d collisionPoint,
            Eigen::Vector3d surfaceNormal )
```

Calculate impact and result of collision with with solid surface. Results are applied to UAV state.

Parameters

COR	coefficient of restitution. e = 0 is perfect inelastic collision, e = 1 is perfect elastic collision. 0 $<$ e $<$ 1 is a real-world inelastic collision, in which some kinetic energy is dissipated.
mi_static	static friction coefficient
mi_dynamic	dynamic friction coefficient
collisionPoint	point of collision
surfaceNormal	surface normal vector

6.3.3.2 calcMomentumConservanceConservation()

```
Eigen::Vector3d Aircraft::calcMomentumConservanceConservation (
            double m,
            Eigen::Vector3d speed,
            Eigen::Vector3d r )  [protected]
```

Calculateds result of releasing/launching object from aircraft.

Parameters

m	object mass
speed	object's initial velocity vector in body frame
r	offset of object

Returns

initial linear velocity of object in world frame

6.3.3.3 dropCargo()

```
std::tuple< int, Eigen::Vector3d > Aircraft::dropCargo (
            int index )
```

Release cargo of specified index.

Parameters

index	index of cargo

Returns

Returns pair of result and velocity of released cargo in world frame. Result is number of cargo of given type that left on board. Result also informs about fails:

- -1 - cooldown, next drop is not ready
- -2 - out of cargos
- -10 - index not found

6.3.3.4 reduceMass()

```
void Aircraft::reduceMass (
            double delta_m,
            Eigen::Vector3d r )  [protected]
```

Reduces mass of aircraft of given value. Mass matrix is reduced proportionally - moments of inertia is scaled as well.

Parameters

delta_m	mass reduction
r	lost mass position in local frame

6.3.3.5 RHS()

```
Eigen::VectorXd Aircraft::RHS (
            double time,
            Eigen::VectorXd local_state )  [protected], [virtual]
```

Right hand side of main differential equation.

Parameters

time	time of simulation
state	state vector

Returns

derivative of state

6.3.3.6 sendState()

```
void Aircraft::sendState (
            zmq::socket_t * socket )
```

Sends simulation state via publisher socket.

Parameters

socket	zmq socket to send simulation state

6.3.3.7 setHinge()

```
bool Aircraft::setHinge (
            char type,
            int index,
            int hinge_index,
            double value )
```

Set angle of specified hinge in specified drive.

Parameters

type	type of drive: 'r' - rotor, 'j' - jet
index	index of drive
hinge_index	index of hinde
value	new angle value

Returns

true, if angle was set. Returns false if specified drive or hinge wasn't found

6.3.3.8 setSurface()

```
bool Aircraft::setSurface (
            Eigen::VectorXd angles )
```

Set surface deflation.

Parameters

angles	vector of new surface angles

Returns

true if angles were set. Returns false if length of vector is not equal to numbers of surfaces

6.3.3.9 shootAmmo()

```
std::tuple< int, Eigen::Vector3d > Aircraft::shootAmmo (
            int index )
```

Shoot ammo of specified index.

Parameters

index	index of ammo

Returns

Returns pair of result and velocity of released cargo in world frame. Result is number of ammo of given type that left on board. Result also informs about fails:

- -1 - cooldown, next shoot is not ready
- -2 - out of ammo
- -10 - index not found

6.3.3.10 startJet()

```
bool Aircraft::startJet (
            int index )
```

Starts jet engine.

Parameters

index	index of engine to start

Returns

return true if jet was started. False if jet is running or burnt out

6.3.3.11 trim()

```
void Aircraft::trim ( )
```

Restore trim values of surface angles.

6.3.3.12 update()

```
void Aircraft::update ( )
```

Simulation step.

6.3.4 Member Data Documentation

6.3.4.1 aero

```
AeroCoefficients Aircraft::aero  [protected]
```

6.3.4.2 ammo

```
std::unique_ptr<Ammo[]> Aircraft::ammo  [protected]
```

6.3.4.3 cargo

```
std::unique_ptr<Cargo[]> Aircraft::cargo  [protected]
```

6.3.4.4 invMassMatrix

```
Matrix<double,6,6> Aircraft::invMassMatrix  [protected]
```

6.3.4.5 jets

```
std::unique_ptr<Jet[]> Aircraft::jets  [protected]
```

6.3.4.6 massMatrix

```
Matrix<double,6,6> Aircraft::massMatrix  [protected]
```

6.3.4.7 mtx

```
std::mutex Aircraft::mtx  [protected]
```

6.3.4.8 noOfAmmo

```
int Aircraft::noOfAmmo  [protected]
```

6.3.4.9 noOfCargo

```
int Aircraft::noOfCargo  [protected]
```

6.3.4.10 noOfJets

```
int Aircraft::noOfJets  [protected]
```

6.3.4.11 noOfRotors

```
int Aircraft::noOfRotors  [protected]
```

6.3.4.12 ode

```
std::unique_ptr<ODE> Aircraft::ode  [protected]
```

6.3.4.13 rotors

```
std::unique_ptr<Rotor[]> Aircraft::rotors  [protected]
```

6.3.4.14 state

```
UAVstate Aircraft::state
```

6.3.4.15 surfaces

```
ControlSurfaces Aircraft::surfaces  [protected]
```

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

- src/aircraft/aircraft.hpp
- src/aircraft/aircraft.cpp
- src/aircraft/aircraft_comm.cpp
- src/aircraft/aircraft_impulse.cpp

6.4 Ammo Class Reference

```
#include <loads.hpp>
```

Inheritance diagram for Ammo:

Collaboration diagram for Ammo:

Public Member Functions

- Ammo ()=default
- Ammo (int ammount, double reload, Eigen::Vector3d offset, double mass, Eigen::Vector3d V0)
- Ammo & operator= (const Ammo &other)
- Eigen::Vector3d getV0 ()

  get start velocity of ammo when launched

Protected Attributes

- Eigen::Vector3d _V0

Additional Inherited Members

6.4.1 Constructor & Destructor Documentation

6.4.1.1 Ammo() [1/2]

```
Ammo::Ammo ( )  [default]
```

6.4.1.2 Ammo() [2/2]

```
Ammo::Ammo (
            int ammount,
            double reload,
            Eigen::Vector3d offset,
            double mass,
            Eigen::Vector3d V0 )
```

6.4.2 Member Function Documentation

6.4.2.1 getV0()

```
Eigen::Vector3d Ammo::getV0 ( )  [inline]
```

get start velocity of ammo when launched

Returns

start velocity vector

6.4.2.2 operator=()

```
Ammo & Ammo::operator= (
            const Ammo & other )
```

6.4.3 Member Data Documentation

6.4.3.1 _V0

```
Eigen::Vector3d Ammo::_V0  [protected]
```

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

- lib/UAV_common/src/components/loads.hpp
- lib/UAV_common/src/components/loads.cpp

6.5 Atmosphere Class Reference

Representation of atmosphere where aircrafts fly.

`#include <atmosphere.hpp>`

Public Member Functions

- Atmosphere ()
  Default constructor.
- ~Atmosphere ()
  Default deconstructor.
- Eigen::Vector3d getWind ()
  Returns wind speed vector in world frame.
- double getAirTemperature ()
  Returns air temperature.
- double getAirPressure ()
  Returns air pressure.
- double getAirDensity ()
  Returns air density.
- void update (AtmosphereInfo info)
  Update atmosphere status.

Static Public Member Functions

- static Atmosphere * getSingleton ()
  Returns pointer to singleton of atmosphere.

6.5.1 Detailed Description

Representation of atmosphere where aircrafts fly.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Atmosphere()

`Atmosphere::Atmosphere ( )`

Default constructor.

6.5.2.2 ~Atmosphere()

```
Atmosphere::~Atmosphere ( )
```

Default deconstructor.

6.5.3 Member Function Documentation

6.5.3.1 getAirDensity()

```
double Atmosphere::getAirDensity ( )
```

Returns air density.

Returns

air density kg/m3

6.5.3.2 getAirPressure()

```
double Atmosphere::getAirPressure ( )
```

Returns air pressure.

Returns

air pressure Pa

6.5.3.3 getAirTemperature()

```
double Atmosphere::getAirTemperature ( )
```

Returns air temperature.

Returns

air temperature K

6.5.3.4 getSingleton()

```
Atmosphere * Atmosphere::getSingleton ( )  [static]
```

Returns pointer to singleton of atmosphere.

Returns

pointer to Atmosphere instance. Nullptr if singleton not exists

6.5.3.5 getWind()

```
Eigen::Vector3d Atmosphere::getWind ( )
```

Returns wind speed vector in world frame.

Returns

wind speed vector m/s

6.5.3.6 update()

```
void Atmosphere::update (
            AtmosphereInfo info )
```

Update atmosphere status.

Parameters

info	dto with new atmosphere info

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

- src/simulation/atmosphere.hpp
- src/simulation/atmosphere.cpp

6.6 AtmosphereInfo Struct Reference

DTO containing atmosphere information.

```
#include <atmosphere.hpp>
```

Public Attributes

- Eigen::Vector3d wind = Eigen::Vector3d(0.0,0.0,0.0)
- double air_temperature = 288.15
- double air_pressure = 101300.0
- double air_density = 1.224

6.6.1 Detailed Description

DTO containing atmosphere information.

6.6.2 Member Data Documentation

6.6.2.1 air_density

```
double AtmosphereInfo::air_density = 1.224
```

6.6.2.2 air_pressure

```
double AtmosphereInfo::air_pressure = 101300.0
```

6.6.2.3 air_temperature

```
double AtmosphereInfo::air_temperature = 288.15
```

6.6.2.4 wind

```
Eigen::Vector3d AtmosphereInfo::wind = Eigen::Vector3d(0.0,0.0,0.0)
```

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

- src/simulation/atmosphere.hpp

6.7 controllers::BangBang Class Reference

`#include <bang_bang.hpp>`

Inheritance diagram for controllers::BangBang:

Collaboration diagram for controllers::BangBang:

Public Member Functions

- BangBang (double high, double low, double delta=0.0)
  Constructor with all Bang-bang controller parameters.
- BangBang (rapidxml::xml_node<> *controller_node)
  Construct controller with parameters from xml.
- double calc (double desired, double actual, [[maybe_unused]] double dt) override
  calc output of controller with specific time step
- void clear () override
  clear internal state
- std::unique_ptr< Controller > clone () const override
  virtual clone method

Additional Inherited Members

6.7.1 Constructor & Destructor Documentation

6.7.1.1 BangBang() [1/2]

```
controllers::BangBang::BangBang (
            double high,
            double low,
            double delta = 0.0 )
```

Constructor with all Bang-bang controller parameters.

Parameters

high	output when error is positive
low	output when error is negative
delta	histeresis symetrical to zero

6.7.1.2 BangBang() [2/2]

```
controllers::BangBang::BangBang (
            rapidxml::xml_node<> * controller_node )
```

Construct controller with parameters from xml.

Parameters

controller_node	xml node with controller params

6.7.2 Member Function Documentation

6.7.2.1 calc()

```
double controllers::BangBang::calc (
            double desired,
            double actual,
            [[maybe_unused] ] double dt )  [override]
```

calc output of controller with specific time step

Parameters

desired	input of controller, desired value
actual	measured actual value
dt	time step

Returns

output of controller

6.7.2.2 clear()

```
void controllers::BangBang::clear ( )  [override], [virtual]
```

clear internal state

Implements Controller.

6.7.2.3 clone()

```
std::unique_ptr< Controller > controllers::BangBang::clone ( ) const  [override], [virtual]
```

virtual clone method

Implements Controller.

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

- lib/UAV_common/src/controllers/impl/bang_bang.hpp
- lib/UAV_common/src/controllers/impl/bang_bang.cpp

6.8 Cargo Class Reference

`#include <loads.hpp>`

Inheritance diagram for Cargo:

Collaboration diagram for Cargo:

Public Member Functions

- Cargo ()=default
- Cargo (int ammount, double reload, Eigen::Vector3d offset, double mass)

Additional Inherited Members

6.8.1 Constructor & Destructor Documentation

6.8.1.1 Cargo() [1/2]

```
Cargo::Cargo ( )  [default]
```

6.8.1.2 Cargo() [2/2]

```
Cargo::Cargo (
            int ammount,
            double reload,
            Eigen::Vector3d offset,
            double mass )
```

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

- lib/UAV_common/src/components/loads.hpp
- lib/UAV_common/src/components/loads.cpp

6.9 Controller Class Reference

`#include <controller.hpp>`

Inheritance diagram for Controller:

Public Member Functions

- Controller ()
  Default constructor.
- ∼Controller ()
  Empty deconstructor for derived classes.
- virtual void set_dt (double dt)
  Set new time step.
- double calc (double desired, double actual)
  calc output of controller
- virtual double calc (double desired, double actual, double dt)=0
  calc output of controller with specific time step
- virtual void clear ()=0
  clear internal state
- virtual std::unique_ptr< Controller > clone () const =0
  virtual clone method

Static Public Member Functions

- static std::unique_ptr< Controller > ControllerFactory (rapidxml::xml_node<> ∗controller_node)
  construct controller from given node. If xml is not valid return nullptr.

Protected Attributes

- double _dt

6.9.1 Constructor & Destructor Documentation

6.9.1.1 Controller()

```
Controller::Controller ( )  [inline]
```

Default constructor.

6.9.1.2 ∼Controller()

```
Controller::~Controller ( )  [inline]
```

Empty deconstructor for derived classes.

6.9.2 Member Function Documentation

6.9.2.1 calc() [1/2]

```
double Controller::calc (
            double desired,
            double actual )  [inline]
```

calc output of controller

Parameters

desired	input of controller, desired value
actual	measured actual value

Returns

output of controller

6.9.2.2 calc() [2/2]

```
virtual double Controller::calc (
            double desired,
            double actual,
            double dt )  [pure virtual]
```

calc output of controller with specific time step

Parameters

desired	input of controller, desired value
actual	measured actual value
dt	time step

Returns

output of controller

Implemented in controllers::PID_Discrete, controllers::PID, and controllers::DoubleSetpoint.

6.9.2.3 clear()

```
virtual void Controller::clear ( )  [pure virtual]
```

clear internal state

Implemented in controllers::ZTransform, controllers::ZTransformStatic< N, D >, controllers::PID_Discrete, controllers::PID, controllers::DoubleSetpoint, and controllers::BangBang.

6.9.2.4 clone()

```
virtual std::unique_ptr<Controller> Controller::clone ( ) const  [pure virtual]
```

virtual clone method

Implemented in controllers::ZTransform, controllers::ZTransformStatic< N, D >, controllers::PID_Discrete, controllers::PID, controllers::DoubleSetpoint, and controllers::BangBang.

6.9.2.5 ControllerFactory()

```
std::unique_ptr< Controller > Controller::ControllerFactory (
            rapidxml::xml_node<> * controller_node )  [static]
```

construct controller from given node. If xml is not valid return nullptr.

Parameters

controller_node	xml node with controller config

6.9.2.6 set_dt()

```
virtual void Controller::set_dt (
            double dt )  [inline], [virtual]
```

Set new time step.

Parameters

dt	new time step

Reimplemented in controllers::PID_Discrete.

6.9.3 Member Data Documentation

6.9.3.1 _dt

```
double Controller::_dt  [protected]
```

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

- lib/UAV_common/src/controllers/controller.hpp
- lib/UAV_common/src/controllers/controller.cpp

6.10 ControllerTest Class Reference

Inheritance diagram for ControllerTest:

Collaboration diagram for ControllerTest:

Protected Member Functions

- void SetUp () override
- void TearDown () override

6.10.1 Member Function Documentation

6.10.1.1 SetUp()

```
void ControllerTest::SetUp ( )  [inline], [override], [protected]
```

6.10.1.2 TearDown()

```
void ControllerTest::TearDown ( )  [inline], [override], [protected]
```

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

- lib/UAV_common/src/controllers/controller_test.cpp

6.11 ControlSurfaces Class Reference

Aircraft's control surfaces.

```
#include <control_surfaces.hpp>
```

Public Member Functions

- ControlSurfaces ()
- ControlSurfaces (int noOfSurfaces, Eigen::Matrix< double, 6,-1 > matrix, Eigen::VectorXd min, Eigen::VectorXd max, Eigen::VectorXd trim)

  Constructor.
- Eigen::Vector< double, 6 > getCoefficients () const
- bool setValues (Eigen::VectorXd new_values)
- void restoreTrim ()
- int getNoOfSurface () const
- Eigen::VectorXd getValues () const

6.11.1 Detailed Description

Aircraft's control surfaces.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 ControlSurfaces() [1/2]

```
ControlSurfaces::ControlSurfaces ( )
```

6.11.2.2 ControlSurfaces() [2/2]

```
ControlSurfaces::ControlSurfaces (
            int noOfSurfaces,
            Eigen::Matrix< double, 6,-1 > matrix,
            Eigen::VectorXd min,
            Eigen::VectorXd max,
            Eigen::VectorXd trim )
```

Constructor.

Parameters

noOfSurfaces	number of independent surfaces
matrix	coefficients matrix
min	vector of min angles
max	vector of max angles
trim	vector of trim angles

6.11.3 Member Function Documentation

6.11.3.1 getCoefficients()

```
Eigen::Vector< double, 6 > ControlSurfaces::getCoefficients ( ) const
```

6.11.3.2 getNoOfSurface()

```
int ControlSurfaces::getNoOfSurface ( ) const  [inline]
```

6.11.3.3 getValues()

```
Eigen::VectorXd ControlSurfaces::getValues ( ) const  [inline]
```

6.11.3.4 restoreTrim()

```
void ControlSurfaces::restoreTrim ( )
```

6.11.3.5 setValues()

```
bool ControlSurfaces::setValues (
            Eigen::VectorXd new_values )
```

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

- lib/UAV_common/src/components/control_surfaces.hpp
- lib/UAV_common/src/components/control_surfaces.cpp

6.12 controllers::DoubleSetpoint Class Reference

```
#include <double_setpoint.hpp>
```

Inheritance diagram for controllers::DoubleSetpoint:

Collaboration diagram for controllers::DoubleSetpoint:

Public Member Functions

- DoubleSetpoint (double high, double mid, double low, double mid_range, double delta=0.0)
  Constructor with all Bang-bang controller parameters.
- DoubleSetpoint (rapidxml::xml_node<> *controller_node)
  Construct controller with parameters from xml.
- double calc (double desired, double actual, double dt) override
  calc output of controller with specific time step
- void clear () override
  clear internal state
- std::unique_ptr< Controller > clone () const override
  virtual clone method

Additional Inherited Members

6.12.1 Constructor & Destructor Documentation

6.12.1.1 DoubleSetpoint() [1/2]

```
controllers::DoubleSetpoint::DoubleSetpoint (
            double high,
            double mid,
            double low,
            double mid_range,
            double delta = 0.0 )
```

Constructor with all Bang-bang controller parameters.

Parameters

high	output when error is in positive range
mid	output when error is in center range
low	output when error is in negative range
mid_range	size of center field from zero
delta	histeresis symetrical to zero

6.12.1.2 DoubleSetpoint() [2/2]

```
controllers::DoubleSetpoint::DoubleSetpoint (
            rapidxml::xml_node<> * controller_node )
```

Construct controller with parameters from xml.

Parameters

controller_node	xml node with controller params

6.12.2 Member Function Documentation

6.12.2.1 calc()

```
double controllers::DoubleSetpoint::calc (
            double desired,
            double actual,
            double dt )  [override], [virtual]
```

calc output of controller with specific time step

Parameters

desired	input of controller, desired value
actual	measured actual value
dt	time step

Returns

output of controller

Implements Controller.

6.12.2.2 clear()

```
void controllers::DoubleSetpoint::clear ( )  [override], [virtual]
```

clear internal state

Implements Controller.

6.12.2.3 clone()

```
std::unique_ptr< Controller > controllers::DoubleSetpoint::clone ( ) const  [override], [virtual]
```

virtual clone method

Implements Controller.

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

- lib/UAV_common/src/controllers/impl/double_setpoint.hpp
- lib/UAV_common/src/controllers/impl/double_setpoint.cpp

6.13 Drive Struct Reference

Drive propelling aircraft.

```
#include <drive.hpp>
```

Inheritance diagram for Drive:

Collaboration diagram for Drive:

Public Attributes

- Eigen::Vector3d position
- Eigen::Vector3d axis
- int noOfHinges
- Hinge hinges [2]

6.13.1 Detailed Description

Drive propelling aircraft.

6.13.2 Member Data Documentation

6.13.2.1 axis

```
Eigen::Vector3d Drive::axis
```

6.13.2.2 hinges

```
Hinge Drive::hinges[2]
```

6.13.2.3 noOfHinges

```
int Drive::noOfHinges
```

6.13.2.4 position

```
Eigen::Vector3d Drive::position
```

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

- lib/UAV_common/src/components/drive.hpp

6.14 EKFScalers Struct Reference

Scalers for EKF.

```
#include <navi.hpp>
```

Public Attributes

- double predictScaler
- double updateScaler
- double baroScaler
- double zScaler

6.14.1 Detailed Description

Scalers for EKF.

6.14.2 Member Data Documentation

6.14.2.1 baroScaler

```
double EKFScalers::baroScaler
```

6.14.2.2 predictScaler

```
double EKFScalers::predictScaler
```

6.14.2.3 updateScaler

```
double EKFScalers::updateScaler
```

6.14.2.4 zScaler

```
double EKFScalers::zScaler
```

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

- lib/UAV_common/src/components/navi.hpp

6.15 Forces Class Reference

```
#include <forces.hpp>
```

Static Public Member Functions

- static Vector< double, 6 > gravity_loads (double mass, const Matrix3d &r_nb)

  Calculates gravity loads acting on UAV.
- static Vector< double, 6 > rotor_lift_loads (int noOfRotors, Rotor *rotors, VectorXd rotorAngularVelocity)

  Calculates loads generated by rotors.
- static Vector< double, 6 > jet_lift_loads (int noOfJets, Jet *jets, double time)

  Calculates loads generated by jet.
- static Vector< double, 6 > aerodynamic_loads (const Vector< double, 6 > &x, Vector3d wind_body, const ControlSurfaces &surface, const AeroCoefficients &aero, double height)

  Calculates aerodynamic loads.
- static VectorXd angularAcceleration (VectorXd demandedAngularVelocity, VectorXd rotorAngularVelocity)

  Calculates acceleration of propellers.
- static void generateCharacteristics (const ControlSurfaces &surface, const AeroCoefficients &aero)

  Generates aerodynamics characteristics and save in csv files.

6.15.1 Member Function Documentation

6.15.1.1 aerodynamic_loads()

```
Vector< double, 6 > Forces::aerodynamic_loads (
            const Vector< double, 6 > & x,
            Vector3d wind_body,
            const ControlSurfaces & surface,
            const AeroCoefficients & aero,
            double height )  [static]
```

Calculates aerodynamic loads.

Parameters

x	vector of UAV velocities
wind_body	vector of wind acting on UAV
surface	reference to ControlSurfaces instance
aero	reference to AeroCoefficients instance
height	absolute height about sea (AMSL)

Returns

loads in body frame

6.15.1.2 angularAcceleration()

```
VectorXd Forces::angularAcceleration (
            VectorXd demandedAngularVelocity,
            VectorXd rotorAngularVelocity )  [static]
```

Calculates acceleration of propellers.

Parameters

demandedAngularVelocity	vector of demanded angular velocities
rotorAngularVelocity	vector of actual angular velocities

Returns

vector of angular accelerations

6.15.1.3 generateCharacteristics()

```
void Forces::generateCharacteristics (
            const ControlSurfaces & surface,
            const AeroCoefficients & aero )  [static]
```

Generates aerodynamics characteristics and save in csv files.

Parameters

surface	reference to ControlSurfaces instance
aero	reference to AeroCoefficients instance

6.15.1.4 gravity_loads()

```
Vector< double, 6 > Forces::gravity_loads (
            double mass,
            const Matrix3d & r_nb )  [static]
```

Calculates gravity loads acting on UAV.

Parameters

mass	aircraft mass
r_nb	rotation matrix from world to body frame

Returns

gravity load in body frame

6.15.1.5 jet_lift_loads()

```
Vector< double, 6 > Forces::jet_lift_loads (
            int noOfJets,
```

```
            Jet * jets,
            double time )  [static]
```

Calculates loads generated by jet.

Parameters

noOfJets	numbers of jets
jets	pointer to jet instance
time	simulation time

Returns

loads in body frame

6.15.1.6 rotor_lift_loads()

```
Vector< double, 6 > Forces::rotor_lift_loads (
            int noOfRotors,
            Rotor * rotors,
            VectorXd rotorAngularVelocity )  [static]
```

Calculates loads generated by rotors.

Parameters

noOfRotors	numbers of rotors
rotors	pointer to rotor instance
rotorAngularVelocity	vector of angular velocities of rotors

Returns

loads in body frame

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

- src/dynamic/forces.hpp
- src/dynamic/forces.cpp

6.16 Hinge Class Reference

Hinge connecting aircraft with drives.

```
#include <hinge.hpp>
```

Public Member Functions

- Hinge ()=default
- Hinge (Eigen::Vector3d axis, double max, double min, double trim)
- Hinge (const Hinge &old)
- Hinge & operator= (const Hinge &old)
- void updateValue (double newValue)

  set new angle on hinge
- const Eigen::Matrix3d getRot ()

  Get rotattion matrix of orientation change due to hinge.

6.16.1 Detailed Description

Hinge connecting aircraft with drives.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Hinge() [1/3]

```
Hinge::Hinge ( )  [default]
```

6.16.2.2 Hinge() [2/3]

```
Hinge::Hinge (
            Eigen::Vector3d axis,
            double max,
            double min,
            double trim )
```

6.16.2.3 Hinge() [3/3]

```
Hinge::Hinge (
            const Hinge & old )
```

6.16.3 Member Function Documentation

6.16.3.1 getRot()

```
const Eigen::Matrix3d Hinge::getRot ( )
```

Get rotattion matrix of orientation change due to hinge.

Returns

rotation matrix

6.16.3.2 operator=()

```
Hinge & Hinge::operator= (
            const Hinge & old )
```

6.16.3.3 updateValue()

```
void Hinge::updateValue (
            double newValue )
```

set new angle on hinge

Parameters

newValue	new angle of hinge

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

- lib/UAV_common/src/components/hinge.hpp
- lib/UAV_common/src/components/hinge.cpp

6.17 Jet Class Reference

Jet rocket engine.

```
#include <drive.hpp>
```

Inheritance diagram for Jet:

Collaboration diagram for Jet:

Public Member Functions

- bool start (double time)

  start jet engine
- double getThrust (double time)

  get thrust in specific time
- double getLastThrust ()

  get last calculated thrust

Public Attributes

- int phases
- Eigen::VectorXd thrust
- Eigen::VectorXd time

6.17.1 Detailed Description

Jet rocket engine.

6.17.2 Member Function Documentation

6.17.2.1 getLastThrust()

```
double Jet::getLastThrust ( )  [inline]
```

get last calculated thrust

Returns

last calculated thrust

6.17.2.2 getThrust()

```
double Jet::getThrust (
            double time )
```

get thrust in specific time

Parameters

time	timestamp

Returns

thrust value in Newtons

6.17.2.3 start()

```
bool Jet::start (
            double time )
```

start jet engine

Parameters

time	timestamp of start

Returns

true if start succesful, false if already started

6.17.3 Member Data Documentation

6.17.3.1 phases

```
int Jet::phases
```

6.17.3.2 thrust

```
Eigen::VectorXd Jet::thrust
```

6.17.3.3 time

```
Eigen::VectorXd Jet::time
```

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

- lib/UAV_common/src/components/drive.hpp
- lib/UAV_common/src/components/drive.cpp

6.18 Load Class Reference

Load of aircraft that can be droped or launched.

`#include <loads.hpp>`

Inheritance diagram for Load:

Public Member Functions

- double getMass ()
  get mass of load
- Eigen::Vector3d getOffset ()
  get offset of load
- int getAmmount ()
  get ammount of load
- int release (double time)
  Try to release load.

Protected Member Functions

- Load ()=default
- Load (int ammount, double reload, Eigen::Vector3d offset, double mass)
- Load & operator= (const Load &other)

6.18.1 Detailed Description

Load of aircraft that can be droped or launched.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Load() [1/2]

```
Load::Load ( )  [protected], [default]
```

6.18.2.2 Load() [2/2]

```
Load::Load (
            int ammount,
            double reload,
            Eigen::Vector3d offset,
            double mass )  [protected]
```

6.18.3 Member Function Documentation

6.18.3.1 getAmmount()

```
int Load::getAmmount ( )  [inline]
```

get ammount of load

Returns

ammount

6.18.3.2 getMass()

```
double Load::getMass ( )  [inline]
```

get mass of load

Returns

mass

6.18.3.3 getOffset()

```
Eigen::Vector3d Load::getOffset ( )  [inline]
```

get offset of load

Returns

offset vector

6.18.3.4 operator=()

```
Load & Load::operator= (
            const Load & other )  [protected]
```

6.18.3.5 release()

```
int Load::release (
            double time )
```

Try to release load.

Parameters

time	

Returns

leftover ammount of loads. Return -1 if load is not ready and -2 if out of load

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

- lib/UAV_common/src/components/loads.hpp
- lib/UAV_common/src/components/loads.cpp

6.19 Logger Class Reference

Log vector data with timestamp in file.

`#include <logger.hpp>`

Public Member Functions

- Logger (std::string path, std::string fmt="", uint8_t group=0)
  Constructor.
- ~Logger ()
  Deconstructor.
- void setFmt (std::string fmt)
  Set new format if was not known in constructor.
- void log (double time, std::initializer_list< Eigen::VectorXd > args)
  Log one row.
- void log (double time, std::initializer_list< double > args)
  Log one row.

Static Public Member Functions

- static void setLogDirectory (std::string subdirectory)
  Set global path that log should be created at. Path will be added to relative path of specific log instance.

6.19.1 Detailed Description

Log vector data with timestamp in file.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 Logger()

```
Logger::Logger (
            std::string path,
            std::string fmt = "",
            uint8_t group = 0 )
```

Constructor.

Parameters

path	relative path with log file name.
fmt	format - information about log structure. First line in log file
group	log group - log will be created only if group is in actual LOGGER_MASK

6.19.2.2 ~Logger()

```
Logger::~Logger ( )
```

Deconstructor.

6.19.3 Member Function Documentation

6.19.3.1 log() [1/2]

```
void Logger::log (
            double time,
            std::initializer_list< double > args )
```

Log one row.

Parameters

time	timestamp
args	list of doubles

6.19.3.2 log() [2/2]

```
void Logger::log (
            double time,
            std::initializer_list< Eigen::VectorXd > args )
```

Log one row.

Parameters

time	timestamp
args	list of double vectors

6.19.3.3 setFmt()

```
void Logger::setFmt (
            std::string fmt )
```

Set new format if was not known in constructor.

Parameters

fmt	new format

6.19.3.4 setLogDirectory()

```
void Logger::setLogDirectory (
            std::string subdirectory )  [static]
```

Set global path that log should be created at. Path will be added to relative path of specific log instance.

Parameters

subdirectory	new global log path

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

- lib/UAV_common/src/logger/logger.hpp
- lib/UAV_common/src/logger/logger.cpp

6.20 Matrices Class Reference

```
#include <matrices.hpp>
```

Static Public Member Functions

- static Matrix< double, 6, 6 > massMatrix ()

  Constucts initial mass matrix, based on parameters.
- static Matrix< double, 6, 6 > gyroMatrix (Vector< double, 6 > x)

  Calculates gyroscopic matrix.
- static Matrix< double, 6, 6 > TMatrix (Vector< double, 6 > y)

  Calculates transformation matrix from body to world frame for velocities.
- static Matrix< double, 6, 6 > TMatrix (Vector< double, 7 > y)

  Calculates transformation matrix from body to world frame for velocities.
- static Matrix< double, 3, 3 > R_nb (const Vector< double, 6 > &y)

Calculates rotation matrix from world to body frame.

- static Matrix< double, 3, 3 > R_nb (const Vector< double, 7 > &y)

  Calculates rotation matrix from world to body frame.
- static Matrix< double, 3, 3 > R_wind_b (double alpha, double beta)

  Calculates rotation matrix from wind to body frame.
- static Vector< double, 6 > quaterionsToRPY (Vector< double, 7 > y)

  Convert position and orientation vector from quaterions to RPY Euler angles.
- static Vector< double, 7 > RPYtoQuaterion (Vector< double, 6 > y)

  Convert position and orientation vector from RPY Euler angles to quaterions.
- static Matrix4d OM_conj (Vector< double, 6 > x)

  Calculates conjugation matrix for angular velocity.
- static Eigen::Matrix3d asSkewSymmeticMatrix (Eigen::Vector3d v)

  Calculate skew symmetic matrix, used in cross products.

6.20.1 Member Function Documentation

6.20.1.1 asSkewSymmeticMatrix()

```
Eigen::Matrix3d Matrices::asSkewSymmeticMatrix (
            Eigen::Vector3d v )  [static]
```

Calculate skew symmetic matrix, used in cross products.

Parameters

v	vector 3d

Returns

skew symmetric matrix

6.20.1.2 gyroMatrix()

```
Matrix< double, 6, 6 > Matrices::gyroMatrix (
            Vector< double, 6 > x )  [static]
```

Calculates gyroscopic matrix.

Parameters

x	velocity vector

Returns

gyroscopic matrix

6.20.1.3 massMatrix()

```
Matrix< double, 6, 6 > Matrices::massMatrix ( )  [static]
```

Constucts initial mass matrix, based on parameters.

Returns

mass matrix

6.20.1.4 OM_conj()

```
Matrix4d Matrices::OM_conj (
            Vector< double, 6 > x )  [static]
```

Calculates conjugation matrix for angular velocity.

Parameters

x	vector of velocites

Returns

conjugation matrix

6.20.1.5 quaterionsToRPY()

```
Vector< double, 6 > Matrices::quaterionsToRPY (
            Vector< double, 7 > y )  [static]
```

Convert position and orientation vector from quaterions to RPY Euler angles.

Parameters

y	position & orientation vector

Returns

position & orientation vector. Orientation is given in RPY

6.20.1.6 R_nb() [1/2]

```
Matrix< double, 3, 3 > Matrices::R_nb (
            const Vector< double, 6 > & y )  [static]
```

Calculates rotation matrix from world to body frame.

Parameters

y	actual position & orietation vector (RPY)

Returns

rotation matrix

6.20.1.7 R_nb() [2/2]

```
Matrix< double, 3, 3 > Matrices::R_nb (
            const Vector< double, 7 > & y )  [static]
```

Calculates rotation matrix from world to body frame.

Parameters

y	actual position & orietation vector (quaterions)

Returns

rotation matrix

6.20.1.8 R_wind_b()

```
Matrix< double, 3, 3 > Matrices::R_wind_b (
            double alpha,
            double beta )  [static]
```

Calculates rotation matrix from wind to body frame.

Parameters

alpha	angle of attack
beta	angle of slide

Returns

rotation matrix

6.20.1.9 RPYtoQuaterion()

```
Vector< double, 7 > Matrices::RPYtoQuaterion (
            Vector< double, 6 > y )  [static]
```

Convert position and orientation vector from RPY Euler angles to quaterions.

Parameters

y	position & orientation vector

Returns

position & orientation vector. Orientation is given in quaterions

6.20.1.10 TMatrix() [1/2]

```
Matrix< double, 6, 6 > Matrices::TMatrix (
            Vector< double, 6 > y )  [static]
```

Calculates transformation matrix from body to world frame for velocities.

Parameters

y	actual position vector

Returns

transformation matrix

6.20.1.11 TMatrix() [2/2]

```
Matrix< double, 6, 6 > Matrices::TMatrix (
            Vector< double, 7 > y )  [static]
```

Calculates transformation matrix from body to world frame for velocities.

Parameters

y	actual position vector, quaterion

Returns

transformation matrix

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

- src/dynamic/matrices.hpp
- src/dynamic/matrices.cpp

6.21 ODE Class Reference

Ordinal differencial equation solver.

`#include <ode.hpp>`

Inheritance diagram for ODE:

Public Types

- enum ODEMethod { Euler , Heun , RK4 , PC2 , PC4 , NONE }

  Supported solving method.

Public Member Functions

- ODE (int micro_steps)

  Constructor.
- virtual ~ODE ()

  Virtual deconstructor.
- virtual Eigen::VectorXd step (double t, Eigen::VectorXd y0, std::function< Eigen::VectorXd(double, Eigen::↩ VectorXd)> rhs_fun, double h)=0

  One step of explicit solving algorithm.
- int getMicrosteps () const

  Return microsteps - number of rhs function calls to calculate on step.

Static Public Member Functions

- static ODEMethod fromString (std::string str)

  Parse solving method from string.
- static std::unique_ptr< ODE > factory (ODEMethod method)

  Factory constructing ODE solvers.
- static int getMicrosteps (ODEMethod method)

  Get microsteps of given method.

6.21.1 Detailed Description

Ordinal differencial equation solver.

6.21.2 Member Enumeration Documentation

6.21.2.1 ODEMethod

```
enum ODE::ODEMethod
```

Supported solving method.

Enumerator

Euler	
Heun	
RK4	
PC2	
PC4	
NONE	

6.21.3 Constructor & Destructor Documentation

6.21.3.1 ODE()

```
ODE::ODE (
            int micro_steps )
```

Constructor.

6.21.3.2 ~ODE()

```
virtual ODE::~ODE ( )  [inline], [virtual]
```

Virtual deconstructor.

6.21.4 Member Function Documentation

6.21.4.1 factory()

```
std::unique_ptr< ODE > ODE::factory (
            ODEMethod method )  [static]
```

Factory constructing ODE solvers.

Parameters

method	type of desired method

Returns

instance of ODE solver

6.21.4.2 fromString()

```
ODE::ODEMethod ODE::fromString (
            std::string str )  [static]
```

Parse solving method from string.

Parameters

str	input string

Returns

solving method if parsed, NONE if unknown

6.21.4.3 getMicrosteps() [1/2]

```
int ODE::getMicrosteps ( ) const
```

Return microsteps - number of rhs function calls to calculate on step.

Returns

microsteps

6.21.4.4 getMicrosteps() [2/2]

```
int ODE::getMicrosteps (
            ODEMethod method )  [static]
```

Get microsteps of given method.

Parameters

method	method type

Returns

number of microstep in one algoritm step

6.21.4.5 step()

```
virtual Eigen::VectorXd ODE::step (
            double t,
            Eigen::VectorXd y0,
            std::function< Eigen::VectorXd(double, Eigen::VectorXd)> rhs_fun,
            double h )  [pure virtual]
```

One step of explicit solving algorithm.

Parameters

t	start time
y0	start variable
rhs_fun	right-hand-side function, calculation of derivative
h	time step

Returns

Implemented in ODE_PC4, ODE_PC2, ODE_RK4, ODE_Heun, and ODE_Euler.

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

- lib/UAV_common/src/ode/ode.hpp
- lib/UAV_common/src/ode/ode.cpp

6.22 ODE_Euler Class Reference

Explicit Euler algorithm.

`#include <ode_impl.hpp>`

Inheritance diagram for ODE_Euler:

Collaboration diagram for ODE_Euler:

Public Member Functions

- ODE_Euler ()
- Eigen::VectorXd step (double t, Eigen::VectorXd y0, std::function< Eigen::VectorXd(double, Eigen::Vector↩ Xd)> rhs_fun, double h) override

  One step of explicit solving algorithm.

Additional Inherited Members

6.22.1 Detailed Description

Explicit Euler algorithm.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 ODE_Euler()

```
ODE_Euler::ODE_Euler ( )  [inline]
```

6.22.3 Member Function Documentation

6.22.3.1 step()

```
Eigen::VectorXd ODE_Euler::step (
            double t,
            Eigen::VectorXd y0,
            std::function< Eigen::VectorXd(double, Eigen::VectorXd)> rhs_fun,
            double h )  [inline], [override], [virtual]
```

One step of explicit solving algorithm.

Parameters

t	start time
y0	start variable
rhs_fun	right-hand-side function, calculation of derivative
h	time step

Returns

Implements ODE.

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

- lib/UAV_common/src/ode/ode_impl.hpp

6.23 ODE_Heun Class Reference

Second order explicit Heun algorithm.

`#include <ode_impl.hpp>`

Inheritance diagram for ODE_Heun:

Collaboration diagram for ODE_Heun:

Public Member Functions

- ODE_Heun ()
- Eigen::VectorXd step (double t, Eigen::VectorXd y0, std::function< Eigen::VectorXd(double, Eigen::Vector↩ Xd)> rhs_fun, double h) override

  One step of explicit solving algorithm.

Additional Inherited Members

6.23.1 Detailed Description

Second order explicit Heun algorithm.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 ODE_Heun()

```
ODE_Heun::ODE_Heun ( )  [inline]
```

6.23.3 Member Function Documentation

6.23.3.1 step()

```
Eigen::VectorXd ODE_Heun::step (
            double t,
            Eigen::VectorXd y0,
            std::function< Eigen::VectorXd(double, Eigen::VectorXd)> rhs_fun,
            double h )  [inline], [override], [virtual]
```

One step of explicit solving algorithm.

Parameters

t	start time
y0	start variable
rhs_fun	right-hand-side function, calculation of derivative
h	time step

Returns

Implements ODE.

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

- lib/UAV_common/src/ode/ode_impl.hpp

6.24 ODE_PC2 Class Reference

Second order predictor-corrector method Second order Adams-bashforth and Adams-moulton.

```
#include <ode_impl.hpp>
```

Inheritance diagram for ODE_PC2:

Collaboration diagram for ODE_PC2:

Public Member Functions

- ODE_PC2 ()
- Eigen::VectorXd step (double t, Eigen::VectorXd y0, std::function< Eigen::VectorXd(double, Eigen::Vector↩ Xd)> rhs_fun, double h) override

  One step of explicit solving algorithm.

Additional Inherited Members

6.24.1 Detailed Description

Second order predictor-corrector method Second order Adams-bashforth and Adams-moulton.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 ODE_PC2()

```
ODE_PC2::ODE_PC2 ( )  [inline]
```

6.24.3 Member Function Documentation

6.24.3.1 step()

```
Eigen::VectorXd ODE_PC2::step (
            double t,
            Eigen::VectorXd y0,
            std::function< Eigen::VectorXd(double, Eigen::VectorXd)> rhs_fun,
            double h )  [inline], [override], [virtual]
```

One step of explicit solving algorithm.

Parameters

t	start time
y0	start variable
rhs_fun	right-hand-side function, calculation of derivative
h	time step

Returns

Implements ODE.

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

- lib/UAV_common/src/ode/ode_impl.hpp

6.25 ODE_PC4 Class Reference

Fourth order predictor-corrector method Fourth order Adams-bashforth and Adams-moulton.

`#include <ode_impl.hpp>`

Inheritance diagram for ODE_PC4:

Collaboration diagram for ODE_PC4:

Public Member Functions

- ODE_PC4 ()
- Eigen::VectorXd step (double t, Eigen::VectorXd y0, std::function< Eigen::VectorXd(double, Eigen::Vector↩ Xd)> rhs_fun, double h) override

  One step of explicit solving algorithm.

Additional Inherited Members

6.25.1 Detailed Description

Fourth order predictor-corrector method Fourth order Adams-bashforth and Adams-moulton.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 ODE_PC4()

```
ODE_PC4::ODE_PC4 ( )  [inline]
```

6.25.3 Member Function Documentation

6.25.3.1 step()

```
Eigen::VectorXd ODE_PC4::step (
            double t,
            Eigen::VectorXd y0,
            std::function< Eigen::VectorXd(double, Eigen::VectorXd)> rhs_fun,
            double h )  [inline], [override], [virtual]
```

One step of explicit solving algorithm.

Parameters

t	start time
y0	start variable
rhs_fun	right-hand-side function, calculation of derivative
h	time step

Returns

Implements ODE.

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

- lib/UAV_common/src/ode/ode_impl.hpp

6.26 ODE_RK4 Class Reference

Fourth order Runge Kutta algorithm.

`#include <ode_impl.hpp>`

Inheritance diagram for ODE_RK4:

Collaboration diagram for ODE_RK4:

Public Member Functions

- ODE_RK4 ()
- Eigen::VectorXd step (double t, Eigen::VectorXd y0, std::function< Eigen::VectorXd(double, Eigen::Vector↩ Xd)> rhs_fun, double h) override

  One step of explicit solving algorithm.

Additional Inherited Members

6.26.1 Detailed Description

Fourth order Runge Kutta algorithm.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 ODE_RK4()

```
ODE_RK4::ODE_RK4 ( )  [inline]
```

6.26.3 Member Function Documentation

6.26.3.1 step()

```
Eigen::VectorXd ODE_RK4::step (
            double t,
            Eigen::VectorXd y0,
            std::function< Eigen::VectorXd(double, Eigen::VectorXd)> rhs_fun,
            double h )  [inline], [override], [virtual]
```

One step of explicit solving algorithm.

Parameters

t	start time
y0	start variable
rhs_fun	right-hand-side function, calculation of derivative
h	time step

Returns

Implements ODE.

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

- lib/UAV_common/src/ode/ode_impl.hpp

6.27 ODETest Class Reference

Inheritance diagram for ODETest:

Collaboration diagram for ODETest:

Protected Member Functions

- void SetUp () override
- void TearDown () override

6.27.1 Member Function Documentation

6.27.1.1 SetUp()

```
void ODETest::SetUp ( )  [inline], [override], [protected]
```

6.27.1.2 TearDown()

```
void ODETest::TearDown ( )  [inline], [override], [protected]
```

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

- lib/UAV_common/src/ode/ode_test.cpp

6.28 Params Class Reference

Simulation parameters.

```
#include <params.hpp>
```

Public Member Functions

- Params ()
  Constructor.
- Params (const Params &)=delete
- Params & operator= (const Params &)=delete
- Params (Params &&)=delete
- ~Params ()
  Deconstructor.

Static Public Member Functions

- static const Params * getSingleton ()
  Get singleton of Params.

Public Attributes

- double STEP_TIME
  Step time of simulation. Step of ODE solving methods.
- std::string ODE_METHOD
  ODE solving method used in simulation.

6.28.1 Detailed Description

Simulation parameters.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 Params() [1/3]

```
Params::Params ( )
```

Constructor.

6.28.2.2 Params() [2/3]

```
Params::Params (
            const Params &  )  [delete]
```

6.28.2.3 Params() [3/3]

```
Params::Params (
            Params &&  )  [delete]
```

6.28.2.4 ~Params()

```
Params::~Params ( )
```

Deconstructor.

6.28.3 Member Function Documentation

6.28.3.1 getSingleton()

```
const Params * Params::getSingleton ( )  [static]
```

Get singleton of Params.

Returns

const pointer to Params instance. Return nullptr if not initialized

6.28.3.2 operator=()

```
Params& Params::operator= (
            const Params &  )  [delete]
```

6.28.4 Member Data Documentation

6.28.4.1 ODE_METHOD

```
std::string Params::ODE_METHOD
```

ODE solving method used in simulation.

6.28.4.2 STEP_TIME

```
double Params::STEP_TIME
```

Step time of simulation. Step of ODE solving methods.

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

- src/params.hpp
- src/params.cpp

6.29 controllers::PID Class Reference

```
#include <PID.hpp>
```

Inheritance diagram for controllers::PID:

Collaboration diagram for controllers::PID:

Public Types

- enum class AntiWindUpMode { NONE , CLAMPING }

  Methods of handling windup in controller.

Public Member Functions

- PID (double Kp, double Ki, double Kd, double Kff=0.0, double min=-std::numeric_limits< double >::max(), double max=std::numeric_limits< double >::max(), AntiWindUpMode antiWindUp=AntiWindUpMode::CLAMPING)

  Constructor with all PID controller parameters.
- PID (rapidxml::xml_node<> *controller_node)

  Construct controller with parameters from xml.
- double calc (double desired, double actual, double dt) override

  calc output of controller with specific time step
- void clear () override

  clear internal state
- std::unique_ptr< Controller > clone () const override

  virtual clone method

Additional Inherited Members

6.29.1 Member Enumeration Documentation

6.29.1.1 AntiWindUpMode

```
enum controllers::PID::AntiWindUpMode  [strong]
```

Methods of handling windup in controller.

Enumerator

NONE	
CLAMPING	

6.29.2 Constructor & Destructor Documentation

6.29.2.1 PID() [1/2]

```
PID::PID (
            double Kp,
            double Ki,
```

```
            double Kd,
            double Kff = 0.0,
            double min = -std::numeric_limits<double>::max(),
            double max = std::numeric_limits<double>::max(),
            AntiWindUpMode antiWindUp = AntiWindUpMode::CLAMPING )
```

Constructor with all PID controller parameters.

Parameters

Kp	P term
Ki	I term
Kd	D term
Kff	FF term
min	saturation - lower range limit
max	saturation - upper range limit
antiWindUp	antiwindup method

6.29.2.2 PID() [2/2]

```
PID::PID (
            rapidxml::xml_node<> * controller_node )
```

Construct controller with parameters from xml.

Parameters

controller_node	xml node with controller params

6.29.3 Member Function Documentation

6.29.3.1 calc()

```
double controllers::PID::calc (
            double desired,
            double actual,
            double dt )  [override], [virtual]
```

calc output of controller with specific time step

Parameters

desired	input of controller, desired value
actual	measured actual value
dt	time step

Returns

output of controller

Implements Controller.

6.29.3.2 clear()

```
void PID::clear ( )  [override], [virtual]
```

clear internal state

Implements Controller.

6.29.3.3 clone()

```
std::unique_ptr< Controller > PID::clone ( ) const  [override], [virtual]
```

virtual clone method

Implements Controller.

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

- lib/UAV_common/src/controllers/impl/PID.hpp
- lib/UAV_common/src/controllers/impl/PID.cpp

6.30 controllers::PID_Discrete Class Reference

```
#include <PID_discrete.hpp>
```

Inheritance diagram for controllers::PID_Discrete:

Collaboration diagram for controllers::PID_Discrete:

Public Member Functions

- PID_Discrete (double Kp, double Ki, double Kd, double Kff=0.0, double N=100.0, double min=-std::numeric↩ _limits< double >::max(), double max=std::numeric_limits< double >::max())
  Constructor with all PID controller parameters.
- PID_Discrete (rapidxml::xml_node<> *controller_node)
  Construct controller with parameters from xml.
- double calc (double desired, double actual, double dt) override
  calc output of controller with specific time step
- void set_dt (double dt) override
  Set new time step.
- void clear () override
  clear internal state
- std::unique_ptr< Controller > clone () const override
  virtual clone method

Additional Inherited Members

6.30.1 Constructor & Destructor Documentation

6.30.1.1 PID_Discrete() [1/2]

```
controllers::PID_Discrete::PID_Discrete (
            double Kp,
            double Ki,
            double Kd,
            double Kff = 0.0,
            double N = 100.0,
            double min = -std::numeric_limits<double>::max(),
            double max = std::numeric_limits<double>::max() )
```

Constructor with all PID controller parameters.

Parameters

Kp	P term
Ki	I term
Kd	D term
Kff	FF term
min	saturation - lower range limit
max	saturation - upper range limit
antiWindUp	antiwindup method

6.30.1.2 PID_Discrete() [2/2]

```
controllers::PID_Discrete::PID_Discrete (
            rapidxml::xml_node<> * controller_node )
```

Construct controller with parameters from xml.

Parameters

controller_node	xml node with controller params

6.30.2 Member Function Documentation

6.30.2.1 calc()

```
double controllers::PID_Discrete::calc (
            double desired,
            double actual,
            double dt )  [override], [virtual]
```

calc output of controller with specific time step

Parameters

desired	input of controller, desired value
actual	measured actual value
dt	time step

Returns

output of controller

Implements Controller.

6.30.2.2 clear()

```
void controllers::PID_Discrete::clear ( )  [override], [virtual]
```

clear internal state

Implements Controller.

6.30.2.3 clone()

```
std::unique_ptr< Controller > controllers::PID_Discrete::clone ( ) const  [override], [virtual]
```

virtual clone method

Implements Controller.

6.30.2.4 set_dt()

```
void controllers::PID_Discrete::set_dt (
            double dt )  [override], [virtual]
```

Set new time step.

Parameters

dt	new time step

Reimplemented from Controller.

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

- lib/UAV_common/src/controllers/impl/PID_discrete.hpp
- lib/UAV_common/src/controllers/impl/PID_discrete.cpp

6.31 Rotor Struct Reference

Rotor engine with controlled speed.

```
#include <drive.hpp>
```

Inheritance diagram for Rotor:

Collaboration diagram for Rotor:

Public Attributes

- double forceCoff
- double torqueCoff
- int direction
- double timeConstant
- double maxSpeed
- double hoverSpeed

6.31.1 Detailed Description

Rotor engine with controlled speed.

6.31.2 Member Data Documentation

6.31.2.1 direction

```
int Rotor::direction
```

6.31.2.2 forceCoff

```
double Rotor::forceCoff
```

6.31.2.3 hoverSpeed

```
double Rotor::hoverSpeed
```

6.31.2.4 maxSpeed

```
double Rotor::maxSpeed
```

6.31.2.5 timeConstant

```
double Rotor::timeConstant
```

6.31.2.6 torqueCoff

```
double Rotor::torqueCoff
```

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

- lib/UAV_common/src/components/drive.hpp

6.32 SensorParams Struct Reference

Base parameters of a sensor.

```
#include <navi.hpp>
```

Public Attributes

- std::string name
- double sd
- Eigen::Vector3d bias
- double refreshTime

6.32.1 Detailed Description

Base parameters of a sensor.

6.32.2 Member Data Documentation

6.32.2.1 bias

```
Eigen::Vector3d SensorParams::bias
```

6.32.2.2 name

```
std::string SensorParams::name
```

6.32.2.3 refreshTime

```
double SensorParams::refreshTime
```

6.32.2.4 sd

```
double SensorParams::sd
```

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

- lib/UAV_common/src/components/navi.hpp

6.33 Simulation Class Reference

```
#include <simulation.hpp>
```

Public Member Functions

- Simulation ()
  Default constructor.
- ~Simulation ()
  Deconstructor.
- void run ()
  Run simulation.

6.33.1 Constructor & Destructor Documentation

6.33.1.1 Simulation()

```
Simulation::Simulation ( )
```

Default constructor.

6.33.1.2 ~Simulation()

```
Simulation::~Simulation ( )
```

Deconstructor.

6.33.2 Member Function Documentation

6.33.2.1 run()

```
void Simulation::run ( )
```

Run simulation.

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

- src/simulation/simulation.hpp
- src/simulation/simulation.cpp

6.34 TimedLoop Class Reference

Simulation of real-time synchronized loop.

```
#include <timed_loop.hpp>
```

Public Member Functions

- TimedLoop (int periodInMs, std::function< void(void)> func, Status &status)
  Constructor.
- void go ()
  start infinite loop
- void go (uint32_t loops)
  start loop for specific cycle numbers

6.34.1 Detailed Description

Simulation of real-time synchronized loop.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 TimedLoop()

```
TimedLoop::TimedLoop (
            int periodInMs,
            std::function< void(void)> func,
            Status & status )
```

Constructor.

Parameters

periodInMs	loop period in milliseconds
func	function that should be called in loop
status	reference to controlling status

6.34.3 Member Function Documentation

6.34.3.1 go() [1/2]

```
void TimedLoop::go ( )
```

start infinite loop

6.34.3.2 go() [2/2]

```
void TimedLoop::go (
            uint32_t loops )
```

start loop for specific cycle numbers

Parameters

loops	how many cycles should be done

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

- lib/UAV_common/src/timed_loop/timed_loop.hpp
- lib/UAV_common/src/timed_loop/timed_loop.cpp

6.35 UAVparams Struct Reference

Parsed UAV configuration from XML.

`#include <uav_params.hpp>`

Collaboration diagram for UAVparams:

Public Member Functions

- UAVparams ()
  Initialize default data.
- ~UAVparams ()
- void loadConfig (std::string configFile)
- Eigen::VectorXd getRotorTimeContants () const
- Eigen::VectorXd getRotorMaxSpeeds () const
- Eigen::VectorXd getRotorHoverSpeeds () const

Static Public Member Functions

- static const UAVparams * getSingleton ()

Public Attributes

- std::string name
- bool instantRun
- std::string initialMode
- Eigen::Vector3d initialPosition
- Eigen::Vector3d initialOrientation
- Eigen::Vector3d initialVelocity
- Eigen::Vector3d target
- double m
- double Ix
- double Iy
- double Iz
- double Ixy
- double Ixz
- double Iyz
- int noOfRotors
- std::unique_ptr< Rotor[ ]> rotors
- int noOfJets
- std::unique_ptr< Jet[ ]> jets
- ControlSurfaces surfaces
- AeroCoefficients aero_coffs

- std::map< std::string, std::unique_ptr< Controller > > controllers
- std::vector< SensorParams > sensors
- AHRSParams ahrs
- EKFScalers ekf
- Eigen::MatrixX4d rotorMixer
- Eigen::MatrixX4d surfaceMixer
- int noOfAmmo
- std::unique_ptr< Ammo[ ]> ammo
- int noOfCargo
- std::unique_ptr< Cargo[ ]> cargo

6.35.1 Detailed Description

Parsed UAV configuration from XML.

6.35.2 Constructor & Destructor Documentation

6.35.2.1 UAVparams()

```
UAVparams::UAVparams ( )
```

Initialize default data.

6.35.2.2 ~UAVparams()

```
UAVparams::~UAVparams ( )
```

6.35.3 Member Function Documentation

6.35.3.1 getRotorHoverSpeeds()

```
Eigen::VectorXd UAVparams::getRotorHoverSpeeds ( ) const
```

6.35.3.2 getRotorMaxSpeeds()

```
Eigen::VectorXd UAVparams::getRotorMaxSpeeds ( ) const
```

6.35.3.3 getRotorTimeContants()

```
Eigen::VectorXd UAVparams::getRotorTimeContants ( ) const
```

6.35.3.4 getSingleton()

```
const UAVparams * UAVparams::getSingleton ( )  [static]
```

6.35.3.5 loadConfig()

```
void UAVparams::loadConfig (
            std::string configFile )
```

6.35.4 Member Data Documentation

6.35.4.1 aero_coffs

```
AeroCoefficients UAVparams::aero_coffs
```

6.35.4.2 ahrs

```
AHRSParams UAVparams::ahrs
```

6.35.4.3 ammo

```
std::unique_ptr<Ammo[]> UAVparams::ammo
```

6.35.4.4 cargo

```
std::unique_ptr<Cargo[]> UAVparams::cargo
```

6.35.4.5 controllers

```
std::map<std::string,std::unique_ptr<Controller> > UAVparams::controllers
```

6.35.4.6 ekf

```
EKFScalers UAVparams::ekf
```

6.35.4.7 initialMode

```
std::string UAVparams::initialMode
```

6.35.4.8 initialOrientation

```
Eigen::Vector3d UAVparams::initialOrientation
```

6.35.4.9 initialPosition

```
Eigen::Vector3d UAVparams::initialPosition
```

6.35.4.10 initialVelocity

```
Eigen::Vector3d UAVparams::initialVelocity
```

6.35.4.11 instantRun

```
bool UAVparams::instantRun
```

6.35.4.12 Ix

```
double UAVparams::Ix
```

6.35.4.13 Ixy

```
double UAVparams::Ixy
```

6.35.4.14 Ixz

```
double UAVparams::Ixz
```

6.35.4.15 Iy

```
double UAVparams::Iy
```

6.35.4.16 Iyz

```
double UAVparams::Iyz
```

6.35.4.17 Iz

```
double UAVparams::Iz
```

6.35.4.18 jets

```
std::unique_ptr<Jet[]> UAVparams::jets
```

6.35.4.19 m

```
double UAVparams::m
```

6.35.4.20 name

```
std::string UAVparams::name
```

6.35.4.21 noOfAmmo

```
int UAVparams::noOfAmmo
```

6.35.4.22 noOfCargo

```
int UAVparams::noOfCargo
```

6.35.4.23 noOfJets

```
int UAVparams::noOfJets
```

6.35.4.24 noOfRotors

```
int UAVparams::noOfRotors
```

6.35.4.25 rotorMixer

```
Eigen::MatrixX4d UAVparams::rotorMixer
```

6.35.4.26 rotors

```
std::unique_ptr<Rotor[]> UAVparams::rotors
```

6.35.4.27 sensors

```
std::vector<SensorParams> UAVparams::sensors
```

6.35.4.28 surfaceMixer

```
Eigen::MatrixX4d UAVparams::surfaceMixer
```

6.35.4.29 surfaces

```
ControlSurfaces UAVparams::surfaces
```

6.35.4.30 target

```
Eigen::Vector3d UAVparams::target
```

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

- lib/UAV_common/src/parser/uav_params.hpp
- lib/UAV_common/src/parser/uav_params.cpp

6.36 UAVstate Struct Reference

```
#include <uav_state.hpp>
```

Public Member Functions

- UAVstate ()
  Default constructor.
- ~UAVstate ()
  Deconstructor.
- Eigen::Vector< double, 7 > getY ()
  Returns position vector Y from state (quaterions)
- Eigen::Vector< double, 6 > getX ()
  Returns velocity vector X.
- Eigen::VectorXd getOm ()
  Returns rotor's angular velocities vector.
- Eigen::VectorXd getDemandedOm ()
  Returns rotor's demanded angular velocities vector.
- Eigen::Vector< double, 6 > getOuterForce ()
  Returns outer force applied to aircraft.
- Eigen::VectorXd getState ()
  Returns raw state vector.
- int getNoOfRotors ()
  Returns number of rotors.
- Eigen::Vector< double, 6 > getAcceleration ()
  Returns aircraft acceleration.
- void setX (Eigen::Vector< double, 6 > newX)
  Set velocity vector X.
- void setDemandedOm (Eigen::VectorXd newOm)
  Set demanded angular velocity vector.
- void setForce (Eigen::Vector3d force, Eigen::Vector3d torque=Eigen::Vector3d(0.0, 0.0, 0.0))
  Set outer load.
- void setAcceleration (Eigen::Vector< double, 6 > newAccel)
  Set acceleration vector.
- UAVstate & operator= (Eigen::VectorXd &other)
  Assigns given raw state vector to state.
- void setStatus (Status newStatus)
  Sets new timed loop status.

Static Public Member Functions

- static void setY (Eigen::VectorXd &state, Eigen::Vector< double, 7 > Y)
  Sets position vector Y in given state (quaterions)
- static Eigen::Vector< double, 7 > getY (const Eigen::VectorXd &state)
  Returns position vector Y from given state (quaterions)
- static void setX (Eigen::VectorXd &state, Eigen::Vector< double, 6 > X)
  Set velocity vector X in given state.
- static void setOm (Eigen::VectorXd &state, Eigen::VectorXd Om)
  Set angular velocity vector in given state.
- static Eigen::Vector< double, 6 > getX (const Eigen::VectorXd &state)
  Returns velocity vector X from given state.
- static Eigen::VectorXd getOm (const Eigen::VectorXd &state)
  Return angular velocity vector from given state.

Public Attributes

- std::atomic< double > real_time
  simulation time
- std::mutex state_mtx
  state mutex
- Status status
  Timed loop status.
- std::condition_variable status_cv

Friends

- std::ostream & operator<< (std::ostream &outs, const UAVstate &state)
  Serializes state to stream.

6.36.1 Constructor & Destructor Documentation

6.36.1.1 UAVstate()

```
UAVstate::UAVstate ( )
```

Default constructor.

6.36.1.2 ~UAVstate()

```
UAVstate::~UAVstate ( )
```

Deconstructor.

6.36.2 Member Function Documentation

6.36.2.1 getAcceleration()

```
Eigen::Vector<double,6> UAVstate::getAcceleration ( )  [inline]
```

Returns aircraft acceleration.

Returns

acceleraton vector

6.36.2.2 getDemandedOm()

```
Eigen::VectorXd UAVstate::getDemandedOm ( )
```

Returns rotor's demanded angular velocities vector.

Returns

rotor's demanded angular velocities vector rad/s

6.36.2.3 getNoOfRotors()

```
int UAVstate::getNoOfRotors ( )  [inline]
```

Returns number of rotors.

Returns

number of rotors

6.36.2.4 getOm() [1/2]

```
Eigen::VectorXd UAVstate::getOm ( )
```

Returns rotor's angular velocities vector.

Returns

rotor's angular velocities vector

6.36.2.5 getOm() [2/2]

```
Eigen::VectorXd UAVstate::getOm (
            const Eigen::VectorXd & state )  [static]
```

Return angular velocity vector from given state.

Parameters

state	source state

Returns

angular velocity vector

6.36.2.6 getOuterForce()

```
Eigen::Vector< double, 6 > UAVstate::getOuterForce ( )
```

Returns outer force applied to aircraft.

Returns

outer force N

6.36.2.7 getState()

```
Eigen::VectorXd UAVstate::getState ( )
```

Returns raw state vector.

Returns

state vector

6.36.2.8 getX() [1/2]

```
Eigen::Vector< double, 6 > UAVstate::getX ( )
```

Returns velocity vector X.

Returns

velocity vector X

6.36.2.9 getX() [2/2]

```
Eigen::Vector< double, 6 > UAVstate::getX (
            const Eigen::VectorXd & state )  [static]
```

Returns velocity vector X from given state.

Parameters

state	source state

Returns

velocity vector X

6.36.2.10 getY() [1/2]

```
Eigen::Vector< double, 7 > UAVstate::getY ( )
```

Returns position vector Y from state (quaterions)

Returns

position vector Y

6.36.2.11 getY() [2/2]

```
Eigen::Vector< double, 7 > UAVstate::getY (
            const Eigen::VectorXd & state )  [static]
```

Returns position vector Y from given state (quaterions)

Parameters

state	source state

Returns

position vector Y

6.36.2.12 operator=()

```
UAVstate & UAVstate::operator= (
            Eigen::VectorXd & other )
```

Assigns given raw state vector to state.

6.36.2.13 setAcceleration()

```
void UAVstate::setAcceleration (
            Eigen::Vector< double, 6 > newAccel )
```

Set acceleration vector.

Parameters

newAccel	new acceleration vector

6.36.2.14 setDemandedOm()

```
void UAVstate::setDemandedOm (
            Eigen::VectorXd newOm )
```

Set demanded angular velocity vector.

Parameters

newOm	new demanded angular velocity vector

6.36.2.15 setForce()

```
void UAVstate::setForce (
            Eigen::Vector3d force,
            Eigen::Vector3d torque = Eigen::Vector3d(0.0,0.0,0.0) )
```

Set outer load.

Parameters

force	applied force
torque	applied torque

6.36.2.16 setOm()

```
void UAVstate::setOm (
            Eigen::VectorXd & state,
            Eigen::VectorXd Om )  [static]
```

Set angular velocity vector in given state.

Parameters

state	state that should be updated
Om	new angular velocity vector

6.36.2.17 setStatus()

```
void UAVstate::setStatus (
            Status newStatus )  [inline]
```

Sets new timed loop status.

Parameters

newStatus	new status

6.36.2.18 setX() [1/2]

```
void UAVstate::setX (
            Eigen::Vector< double, 6 > newX )
```

Set velocity vector X.

Parameters

newX	new velocity vector X

6.36.2.19 setX() [2/2]

```
void UAVstate::setX (
            Eigen::VectorXd & state,
            Eigen::Vector< double, 6 > X )  [static]
```

Set velocity vector X in given state.

Parameters

state	state that should be updated
X	new velocity vector X

6.36.2.20 setY()

```
void UAVstate::setY (
            Eigen::VectorXd & state,
            Eigen::Vector< double, 7 > Y )  [static]
```

Sets position vector Y in given state (quaterions)

Parameters

state	state that should be updated
Y	new position vector

6.36.3 Friends And Related Function Documentation

6.36.3.1 operator<<

```
std::ostream& operator<< (
            std::ostream & outs,
            const UAVstate & state )  [friend]
```

Serializes state to stream.

6.36.4 Member Data Documentation

6.36.4.1 real_time

```
std::atomic<double> UAVstate::real_time
```

simulation time

6.36.4.2 state_mtx

```
std::mutex UAVstate::state_mtx
```

state mutex

6.36.4.3 status

```
Status UAVstate::status
```

Timed loop status.

6.36.4.4 status_cv

```
std::condition_variable UAVstate::status_cv
```

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

- src/simulation/uav_state.hpp
- src/simulation/uav_state.cpp

6.37 controllers::ZTransform Class Reference

```
#include <z_trans.hpp>
```

Inheritance diagram for controllers::ZTransform:

Collaboration diagram for controllers::ZTransform:

Public Member Functions

- ZTransform (const std::vector< double > &num, const std::vector< double > &den, double min=-std::numeric_limits< double >::max(), double max=std::numeric_limits< double >::max())
  Constructorof Z-Transform controller.
- ZTransform (rapidxml::xml_node<> *controller_node)
  Construct controller with parameters from xml.
- double calc (double desired, double actual, [[maybe_unused]] double dt) override
  calc output of controller
- void clear () override
  clear internal state
- std::unique_ptr< Controller > clone () const override
  virtual clone method

Additional Inherited Members

6.37.1 Constructor & Destructor Documentation

6.37.1.1 ZTransform() [1/2]

```
ZTransform::ZTransform (
            const std::vector< double > & num,
            const std::vector< double > & den,
            double min = -std::numeric_limits<double>::max(),
            double max = std::numeric_limits<double>::max() )
```

Constructorof Z-Transform controller.

Parameters

min	saturation - lower range limit
max	saturation - upper range limit

6.37.1.2 ZTransform() [2/2]

```
controllers::ZTransform::ZTransform (
            rapidxml::xml_node<> * controller_node )
```

Construct controller with parameters from xml.

Parameters

controller_node	xml node with controller params

6.37.2 Member Function Documentation

6.37.2.1 calc()

```
double ZTransform::calc (
            double desired,
            double actual,
            [[maybe_unused] ] double dt )  [override]
```

calc output of controller

Parameters

desired	input of controller, desired value
actual	measured actual value

Returns

output of controller

6.37.2.2 clear()

```
void ZTransform::clear ( )  [override], [virtual]
```

clear internal state

Implements Controller.

6.37.2.3 clone()

```
std::unique_ptr< Controller > ZTransform::clone ( ) const  [override], [virtual]
```

virtual clone method

Implements Controller.

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

- lib/UAV_common/src/controllers/impl/z_trans.hpp
- lib/UAV_common/src/controllers/impl/z_trans.cpp

6.38 controllers::ZTransformStatic< N, D > Class Template Reference

```
#include <z_trans.hpp>
```

Inheritance diagram for controllers::ZTransformStatic< N, D >:

Collaboration diagram for controllers::ZTransformStatic< N, D >:

Public Member Functions

- ZTransformStatic (const std::array< double, N > &num, const std::array< double, D > &den, double min=-std::numeric_limits< double >::max(), double max=std::numeric_limits< double >::max())
  Constructorof Z-Transform controller.
- ZTransformStatic (rapidxml::xml_node<> *controller_node)=delete
  Construct controller with parameters from xml.
- double calc (double desired, double actual, [[maybe_unused]] double dt) override
  calc output of controller
- void clear () override
  clear internal state
- std::unique_ptr< Controller > clone () const override
  virtual clone method

Additional Inherited Members

6.38.1 Constructor & Destructor Documentation

6.38.1.1 ZTransformStatic() [1/2]

```
template<unsigned int N, unsigned int D>
controllers::ZTransformStatic< N, D >::ZTransformStatic (
            const std::array< double, N > & num,
            const std::array< double, D > & den,
            double min = -std::numeric_limits<double>::max(),
            double max = std::numeric_limits<double>::max() )
```

Constructorof Z-Transform controller.

Parameters

min	saturation - lower range limit
max	saturation - upper range limit

6.38.1.2 ZTransformStatic() [2/2]

```
template<unsigned int N, unsigned int D>
controllers::ZTransformStatic< N, D >::ZTransformStatic (
            rapidxml::xml_node<> * controller_node )  [delete]
```

Construct controller with parameters from xml.

Parameters

controller_node	xml node with controller params

6.38.2 Member Function Documentation

6.38.2.1 calc()

```
template<unsigned int N, unsigned int D>
double controllers::ZTransformStatic< N, D >::calc (
            double desired,
            double actual,
            [[maybe_unused] ] double dt )  [override]
```

calc output of controller

Parameters

desired	input of controller, desired value
actual	measured actual value

Returns

output of controller

6.38.2.2 clear()

```
template<unsigned int N, unsigned int D>
void controllers::ZTransformStatic< N, D >::clear  [override], [virtual]
```

clear internal state

Implements Controller.

6.38.2.3 clone()

```
template<unsigned int N, unsigned int D>
std::unique_ptr< Controller > controllers::ZTransformStatic< N, D >::clone  [override], [virtual]
```

virtual clone method

Implements Controller.

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

- lib/UAV_common/src/controllers/impl/z_trans.hpp

Chapter 7

File Documentation

7.1 build/CMakeFiles/3.22.1/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_VERSION

Functions

- int main (int argc, char *argv[ ])

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

7.1.1 Macro Definition Documentation

7.1.1.1 __has_include

```
#define __has_include(
              x ) 0
```

7.1.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

7.1.1.3 C_VERSION

```
#define C_VERSION
```

7.1.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

7.1.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

7.1.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

7.1.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

7.1.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

7.1.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

7.1.2 Function Documentation

7.1.2.1 main()

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

7.1.3 Variable Documentation

7.1.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

7.1.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

7.1.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

7.1.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=
  "INFO" ":" "standard_default[" C_VERSION "]"
```

7.1.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

7.2 build/CMakeFiles/3.22.1/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD __cplusplus

Functions

- int main (int argc, char *argv[])

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

7.2.1 Macro Definition Documentation

7.2.1.1 __has_include

```
#define __has_include(
              x ) 0
```

7.2.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

7.2.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

7.2.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

7.2.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

7.2.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

7.2.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

7.2.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

7.2.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

7.2.2 Function Documentation

7.2.2.1 main()

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

7.2.3 Variable Documentation

7.2.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

7.2.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

7.2.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

7.2.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["
  "98"
"]"
```

7.2.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

```
#include "../src/ode/ode.hpp"
#include "../src/controllers/controller.hpp"
#include "../src/timed_loop/timed_loop.hpp"
#include "../src/timed_loop/status.hpp"
#include "../src/parser/parser.hpp"
#include "../src/parser/uav_params.hpp"
#include "../src/components/components.hpp"
```

Include dependency graph for common.hpp: This graph shows which files directly or indirectly include this file:

7.31 lib/UAV_common/scripts/controller_plots.m File Reference

Functions

- plot (x, y, 'DisplayName', csvFiles(i).name)
- end xlabel ('Czas')
- ylabel ('Wartość regulowana')
- title ('Test regulatorów')
- legend ('Location', 'Best')

Variables

- clc
- clear folderPath = '../build/controller_plots/'
- csvFiles = dir(fullfile(folderPath, '∗.csv'))
- figure
- hold on
- for i
- data = readmatrix(filePath)
- x = data(:, 1)
- y = data(:, 2)
- hold off

7.31.1 Function Documentation

7.31.1.1 legend()

```
legend (
            'Location' ,
            'Best'  )
```

7.31.1.2 plot()

```
plot (
            x ,
            y ,
            'DisplayName' ,
            csvFiles(i).  name )
```

7.31.1.3 title()

```
title (
            'Test regulatorów'  )
```

7.31.1.4 xlabel()

```
end xlabel (
            'Czas'  )
```

7.31.1.5 ylabel()

```
ylabel (
            'Wartość regulowana'  )
```

7.31.2 Variable Documentation

7.31.2.1 clc

```
clc
```

7.31.2.2 csvFiles

```
csvFiles = dir(fullfile(folderPath, '*.csv'))
```

7.31.2.3 data

```
data = readmatrix(filePath)
```

7.31.2.4 figure

```
figure
```

7.31.2.5 folderPath

```
clear folderPath = '../build/controller_plots/'
```

7.31.2.6 i

```
for i
```

Initial value:

```
= 1:length(csvFiles)
    filePath = fullfile(folderPath, csvFiles(i).name)
```

7.31.2.7 off

```
hold off
```

7.31.2.8 on

```
hold on
```

7.31.2.9 x

```
x = data(:, 1)
```

7.31.2.10 y

```
y = data(:, 2)
```

7.32 lib/UAV_common/src/components/aero_coefficients.hpp File Reference

```
#include <Eigen/Dense>
```
Include dependency graph for aero_coefficients.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct AeroCoefficients

  Aerodynamic coefficient.

7.33 lib/UAV_common/src/components/components.hpp File Reference

```
#include "drive.hpp"
#include "control_surfaces.hpp"
#include "aero_coefficients.hpp"
#include "loads.hpp"
#include "navi.hpp"
```
Include dependency graph for components.hpp: This graph shows which files directly or indirectly include this file:

7.34 lib/UAV_common/src/components/control_surfaces.cpp File Reference

```
#include "control_surfaces.hpp"
```
Include dependency graph for control_surfaces.cpp:

7.35 lib/UAV_common/src/components/control_surfaces.hpp File Reference

```
#include <Eigen/Dense>
```
Include dependency graph for control_surfaces.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ControlSurfaces

  Aircraft's control surfaces.

7.36 lib/UAV_common/src/components/drive.cpp File Reference

```
#include "drive.hpp"
```
Include dependency graph for drive.cpp:

7.37 lib/UAV_common/src/components/drive.hpp File Reference

```
#include <Eigen/Dense>
#include "hinge.hpp"
```
Include dependency graph for drive.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct Drive

  Drive propelling aircraft.
- struct Rotor

  Rotor engine with controlled speed.
- class Jet

  Jet rocket engine.

7.38 lib/UAV_common/src/components/hinge.cpp File Reference

```
#include "hinge.hpp"
```
Include dependency graph for hinge.cpp:

Functions

- Eigen::Matrix3d asSkewMatrix (Eigen::Vector3d v)

7.38.1 Function Documentation

7.38.1.1 asSkewMatrix()

```
Eigen::Matrix3d asSkewMatrix (
            Eigen::Vector3d v )
```

7.39 lib/UAV_common/src/components/hinge.hpp File Reference

```
#include <Eigen/Dense>
#include <mutex>
#include <memory>
```
Include dependency graph for hinge.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Hinge

  Hinge connecting aircraft with drives.

7.40 lib/UAV_common/src/components/loads.cpp File Reference

```
#include "loads.hpp"
#include <limits>
```

Include dependency graph for loads.cpp:

7.41 lib/UAV_common/src/components/loads.hpp File Reference

```
#include <Eigen/Dense>
#include <atomic>
```

Include dependency graph for loads.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Load

  Load of aircraft that can be droped or launched.
- class Ammo
- class Cargo

7.42 lib/UAV_common/src/components/navi.hpp File Reference

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

Include dependency graph for navi.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct SensorParams

  Base parameters of a sensor.
- struct AHRSParams

  AHRS parameters.
- struct EKFScalers

  Scalers for EKF.

7.43 lib/UAV_common/src/controllers/controller.cpp File Reference

```
#include "controller.hpp"
#include "impl/PID.hpp"
#include "impl/PID_discrete.hpp"
#include "impl/bang_bang.hpp"
#include "impl/double_setpoint.hpp"
#include "impl/z_trans.hpp"
#include <cstring>
#include <stdexcept>
```

Include dependency graph for controller.cpp:

7.44 lib/UAV_common/src/controllers/controller.hpp File Reference

```
#include <memory>
#include "rapidxml/rapidxml.hpp"
```

Include dependency graph for controller.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Controller

7.45 lib/UAV_common/src/controllers/controller_test.cpp File Reference

```
#include "impl/PID.hpp"
#include "impl/PID_discrete.hpp"
#include "impl/bang_bang.hpp"
#include "impl/double_setpoint.hpp"
#include "impl/z_trans.hpp"
#include <gtest/gtest.h>
#include <memory>
#include <filesystem>
#include <fstream>
```

Include dependency graph for controller_test.cpp:

Classes

- class ControllerTest

Functions

- std::vector< std::shared_ptr< Controller > > getMethodsToTest ()
- TEST_P (ControllerTest, TestConstFunction)
- TEST_P (ControllerTest, SimpleObjectControl)
- INSTANTIATE_TEST_SUITE_P (TestDerivedClasses, ControllerTest, testing::ValuesIn(getMethodsToTest()))
- int main (int argc, char **argv)

Variables

- constexpr bool plot = true
- constexpr auto plot_directory_name = "controller_plots"

7.45.1 Function Documentation

7.45.1.1 getMethodsToTest()

```
std::vector<std::shared_ptr<Controller> > getMethodsToTest ( )
```

7.45.1.2 INSTANTIATE_TEST_SUITE_P()

```
INSTANTIATE_TEST_SUITE_P (
            TestDerivedClasses ,
            ControllerTest ,
            testing::ValuesIn(getMethodsToTest())  )
```

7.45.1.3 main()

```
int main (
            int argc,
            char ** argv )
```

7.45.1.4 TEST_P() [1/2]

```
TEST_P (
            ControllerTest ,
            SimpleObjectControl  )
```

7.45.1.5 TEST_P() [2/2]

```
TEST_P (
            ControllerTest ,
            TestConstFunction  )
```

7.45.2 Variable Documentation

7.45.2.1 plot

```
constexpr bool plot = true  [constexpr]
```

7.45.2.2 plot_directory_name

```
constexpr auto plot_directory_name = "controller_plots"  [constexpr]
```

7.46 lib/UAV_common/src/controllers/impl/bang_bang.cpp File Reference

```
#include "bang_bang.hpp"
#include <cstring>
#include <string>
```
Include dependency graph for bang_bang.cpp:

7.47 lib/UAV_common/src/controllers/impl/bang_bang.hpp File Reference

```
#include <memory>
#include "rapidxml/rapidxml.hpp"
#include "../controller.hpp"
```
Include dependency graph for bang_bang.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class controllers::BangBang

Namespaces

- controllers

7.48 lib/UAV_common/src/controllers/impl/double_setpoint.cpp File Reference

```
#include "double_setpoint.hpp"
#include <cstring>
#include <string>
```
Include dependency graph for double_setpoint.cpp:

7.49 lib/UAV_common/src/controllers/impl/double_setpoint.hpp File Reference

```
#include <memory>
#include "rapidxml/rapidxml.hpp"
#include "../controller.hpp"
```
Include dependency graph for double_setpoint.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class controllers::DoubleSetpoint

Namespaces

- controllers

7.50 lib/UAV_common/src/controllers/impl/PID.cpp File Reference

```
#include "PID.hpp"
#include <algorithm>
#include <cstring>
#include <string>
#include <stdexcept>
```

Include dependency graph for PID.cpp:

7.51 lib/UAV_common/src/controllers/impl/PID.hpp File Reference

```
#include <memory>
#include <limits>
#include "rapidxml/rapidxml.hpp"
#include "../controller.hpp"
#include "z_trans.hpp"
```

Include dependency graph for PID.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class controllers::PID

Namespaces

- controllers

7.52 lib/UAV_common/src/controllers/impl/PID_discrete.cpp File Reference

```
#include "PID_discrete.hpp"
#include <iostream>
#include <string>
#include <cstring>
```

Include dependency graph for PID_discrete.cpp:

7.53 lib/UAV_common/src/controllers/impl/PID_discrete.hpp File Reference

```
#include <memory>
#include <limits>
#include "rapidxml/rapidxml.hpp"
#include "../controller.hpp"
#include "z_trans.hpp"
```

Include dependency graph for PID_discrete.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class controllers::PID_Discrete

Namespaces

- controllers

7.54 lib/UAV_common/src/controllers/impl/z_trans.cpp File Reference

```
#include "z_trans.hpp"
#include <sstream>
#include <iterator>
#include <string>
#include <cstring>
```

Include dependency graph for z_trans.cpp:

Functions

- std::vector< double > splitStringToDoubleVector (const std::string &input)

7.54.1 Function Documentation

7.54.1.1 splitStringToDoubleVector()

```
std::vector<double> splitStringToDoubleVector (
            const std::string & input )
```

7.55 lib/UAV_common/src/controllers/impl/z_trans.hpp File Reference

```
#include <memory>
#include <limits>
#include <array>
#include <vector>
#include <algorithm>
#include <numeric>
#include <stdexcept>
#include <cassert>
#include "rapidxml/rapidxml.hpp"
#include "../controller.hpp"
```

Include dependency graph for z_trans.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class controllers::ZTransformStatic< N, D >
- class controllers::ZTransform

Namespaces

- controllers

7.56 lib/UAV_common/src/logger/logger.cpp File Reference

```
#include "logger.hpp"
#include <Eigen/Dense>
#include <iostream>
#include <fstream>
#include <initializer_list>
#include <string>
#include <filesystem>
```

Include dependency graph for logger.cpp:

Functions

- bool shouldLog (uint8_t group)

7.56.1 Function Documentation

7.56.1.1 shouldLog()

```
bool shouldLog (
            uint8_t group )
```

7.57 lib/UAV_common/src/logger/logger.hpp File Reference

```
#include <Eigen/Dense>
#include <iostream>
#include <fstream>
#include <initializer_list>
#include <string>
#include <filesystem>
```

Include dependency graph for logger.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Logger

  Log vector data with timestamp in file.

Macros

- #define LOGGER_MASK -1

7.57.1 Macro Definition Documentation

7.57.1.1 LOGGER_MASK

```
#define LOGGER_MASK -1
```

7.58 lib/UAV_common/src/ode/ode.cpp File Reference

```
#include "ode.hpp"
#include "ode_impl.hpp"
```

Include dependency graph for ode.cpp:

7.59 lib/UAV_common/src/ode/ode.hpp File Reference

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

Include dependency graph for ode.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ODE

  Ordinal differencial equation solver.

7.60 lib/UAV_common/src/ode/ode_impl.hpp File Reference

```
#include "ode.hpp"
```
Include dependency graph for ode_impl.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ODE_Euler

  Explicit Euler algorithm.
- class ODE_Heun

  Second order explicit Heun algorithm.
- class ODE_RK4

  Fourth order Runge Kutta algorithm.
- class ODE_PC2

  Second order predictor-corrector method Second order Adams-bashforth and Adams-moulton.
- class ODE_PC4

  Fourth order predictor-corrector method Fourth order Adams-bashforth and Adams-moulton.

7.61 lib/UAV_common/src/ode/ode_test.cpp File Reference

```
#include "ode.hpp"
#include <gtest/gtest.h>
#include <numbers>
```
Include dependency graph for ode_test.cpp:

Classes

- class ODETest

Functions

- std::vector< ODE::ODEMethod > getMethodsToTest ()
- TEST_F (ODETest, FromStringTest)
- TEST_F (ODETest, FactoryTest)
- TEST_P (ODETest, TestConstFunction)
- TEST_P (ODETest, TestFirstOrder)
- TEST_P (ODETest, TestRHSCalls)
- TEST_P (ODETest, TestHarmonicOscillator)
- INSTANTIATE_TEST_SUITE_P (TestDerivedClasses, ODETest, testing::ValuesIn(getMethodsToTest()))
- int main (int argc, char **argv)

7.61.1 Function Documentation

7.61.1.1 getMethodsToTest()

```
std::vector<ODE::ODEMethod> getMethodsToTest ( )
```

7.61.1.2 INSTANTIATE_TEST_SUITE_P()

```
INSTANTIATE_TEST_SUITE_P (
            TestDerivedClasses ,
            ODETest ,
            testing::ValuesIn(getMethodsToTest())  )
```

7.61.1.3 main()

```
int main (
            int argc,
            char ** argv )
```

7.61.1.4 TEST_F() [1/2]

```
TEST_F (
            ODETest ,
            FactoryTest  )
```

7.61.1.5 TEST_F() [2/2]

```
TEST_F (
            ODETest ,
            FromStringTest  )
```

7.61.1.6 TEST_P() [1/4]

```
TEST_P (
            ODETest ,
            TestConstFunction  )
```

7.61.1.7 TEST_P() [2/4]

```
TEST_P (
             ODETest ,
             TestFirstOrder  )
```

7.61.1.8 TEST_P() [3/4]

```
TEST_P (
             ODETest ,
             TestHarmonicOscillator  )
```

7.61.1.9 TEST_P() [4/4]

```
TEST_P (
             ODETest ,
             TestRHSCalls  )
```

7.62 lib/UAV_common/src/parser/parser.cpp File Reference

```
#include "parser.hpp"
#include <Eigen/Dense>
#include <iostream>
#include <sstream>
```

Include dependency graph for parser.cpp:

Functions

- Eigen::MatrixXd parseMatrixXd (const std::string &input, int R, int C, char delimiter)
  Parse input string to double matrix of specific shape and delimiter.
- Eigen::VectorXd parseVectorXd (std::string str, int noOfElem, char delimiter)
  Parse input string to double vector of specific length and delimiter.

7.62.1 Function Documentation

7.62.1.1 parseMatrixXd()

```
Eigen::MatrixXd parseMatrixXd (
             const std::string & input,
             int R,
             int C,
             char delimiter = ' ' )
```

Parse input string to double matrix of specific shape and delimiter.

Parameters

input	input string
R	number of rows
C	number of columns
delimiter	delimiter

Returns

parsed matrix

7.62.1.2 parseVectorXd()

```
Eigen::VectorXd parseVectorXd (
            std::string str,
            int noOfElem,
            char delimiter = ' ' )
```

Parse input string to double vector of specific length and delimiter.

Parameters

str	input string
noOfElem	length of vector
delimiter	delimiter

Returns

parsed vector

7.63 lib/UAV_common/src/parser/parser.hpp File Reference

`#include <Eigen/Dense>`
Include dependency graph for parser.hpp: This graph shows which files directly or indirectly include this file:

Functions

- Eigen::MatrixXd parseMatrixXd (const std::string &input, int R, int C, char delimiter=' ')
  Parse input string to double matrix of specific shape and delimiter.
- Eigen::VectorXd parseVectorXd (std::string str, int noOfElem, char delimiter=' ')
  Parse input string to double vector of specific length and delimiter.

7.63.1 Function Documentation

7.63.1.1 parseMatrixXd()

```
Eigen::MatrixXd parseMatrixXd (
            const std::string & input,
            int R,
            int C,
            char delimiter = ' ' )
```

Parse input string to double matrix of specific shape and delimiter.

Parameters

input	input string
R	number of rows
C	number of columns
delimiter	delimiter

Returns

parsed matrix

7.63.1.2 parseVectorXd()

```
Eigen::VectorXd parseVectorXd (
            std::string str,
            int noOfElem,
            char delimiter = ' ' )
```

Parse input string to double vector of specific length and delimiter.

Parameters

str	input string
noOfElem	length of vector
delimiter	delimiter

Returns

parsed vector

7.64 lib/UAV_common/src/parser/uav_params.cpp File Reference

```
#include <Eigen/Dense>
#include "uav_params.hpp"
#include <iostream>
#include <fstream>
#include <filesystem>
```

```
#include <mutex>
#include "rapidxml/rapidxml.hpp"
#include "parser.hpp"
```
Include dependency graph for uav_params.cpp:

Functions

- void parseHinge (rapidxml::xml_node<> *hingeNode, Hinge *hinge)

7.64.1 Function Documentation

7.64.1.1 parseHinge()

```
void parseHinge (
            rapidxml::xml_node<> * hingeNode,
            Hinge * hinge )
```

7.65 lib/UAV_common/src/parser/uav_params.hpp File Reference

```
#include <Eigen/Dense>
#include <mutex>
#include <memory>
#include <map>
#include "rapidxml/rapidxml.hpp"
#include "../components/components.hpp"
#include "../controllers/controller.hpp"
```
Include dependency graph for uav_params.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct UAVparams

  Parsed UAV configuration from XML.

7.66 lib/UAV_common/src/timed_loop/status.hpp File Reference

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

Enumerations

- enum Status { idle = 1 , running = 2 , exiting = 3 , reload = 4 }

  status of timed loop. Control it's job

7.66.1 Enumeration Type Documentation

7.66.1.1 Status

```
enum Status
```

status of timed loop. Control it's job

Enumerator

idle	loop is ready to run
running	loop is running
exiting	loop will be break in next occasion.
reload	loop job should be reloaded

7.67 lib/UAV_common/src/timed_loop/timed_loop.cpp File Reference

```
#include "timed_loop.hpp"
#include <stdint.h>
#include <chrono>
#include <thread>
#include "status.hpp"
#include <iostream>
```

Include dependency graph for timed_loop.cpp:

7.68 lib/UAV_common/src/timed_loop/timed_loop.hpp File Reference

```
#include <stdint.h>
#include <functional>
#include "status.hpp"
```

Include dependency graph for timed_loop.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class TimedLoop

  Simulation of real-time synchronized loop.

7.69 src/aircraft/aircraft.cpp File Reference

```
#include "aircraft.hpp"
#include "../params.hpp"
```

Include dependency graph for aircraft.cpp:

Functions

- void clampOrientationIfNessessery ([[maybe_unused]] Eigen::VectorXd &state)

7.69.1 Function Documentation

7.69.1.1 clampOrientationIfNessessery()

```
void clampOrientationIfNessessery (
            [[maybe_unused] ] Eigen::VectorXd & state )
```

7.70 src/aircraft/aircraft.hpp File Reference

```
#include <Eigen/Dense>
#include <memory>
#include <mutex>
#include <zmq.hpp>
#include <iostream>
#include "common.hpp"
#include "../defines.hpp"
#include "../dynamic/forces.hpp"
#include "../dynamic/matrices.hpp"
#include "../simulation/atmosphere.hpp"
#include "../simulation/uav_state.hpp"
```

Include dependency graph for aircraft.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Aircraft

  central class in simulation

7.71 src/aircraft/aircraft_comm.cpp File Reference

```
#include "aircraft.hpp"
```

Include dependency graph for aircraft_comm.cpp:

7.72 src/aircraft/aircraft_impulse.cpp File Reference

```
#include "aircraft.hpp"
```

Include dependency graph for aircraft_impulse.cpp:

7.73 src/defines.hpp File Reference

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

Namespaces

- def

  Simulation constants.

Macros

- #define USE_QUATERIONS 1

  define to use quaterion instead of RPY angles

Variables

- const double def::GRAVITY_CONST = 9.81

  Gravity constant on Earth in m/s2.
- const double def::FRICTION_EPS = 0.001

  minimal friction that is calculated (numerical float eps)
- const double def::GENTLY_PUSH = 0.15

  artificial force coefficient. Protect again diving objects in horizontal wall
- const double def::DOUBLE_EPS = 1e-5

  near zero floating point eps
- const double def::MIXING_FUNCTION = 0.1

  mixing window used in blending normal coefficients with standard ones, when stall angle was exceeded
- const int def::validityOfForce = 5

  how many times outer force should be used

7.73.1 Macro Definition Documentation

7.73.1.1 USE_QUATERIONS

```
#define USE_QUATERIONS 1
```

define to use quaterion instead of RPY angles

7.74 src/dynamic/forces.cpp File Reference

```
#include <Eigen/Dense>
#include <cmath>
#include <iostream>
#include <numbers>
#include "forces.hpp"
#include "matrices.hpp"
#include "../defines.hpp"
#include "../simulation/atmosphere.hpp"
#include "common.hpp"
```

Include dependency graph for forces.cpp:

7.75 src/dynamic/forces.hpp File Reference

```
#include <Eigen/Dense>
#include "common.hpp"
#include "matrices.hpp"
```

Include dependency graph for forces.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Forces

7.76 src/dynamic/matrices.cpp File Reference

```
#include <Eigen/Dense>
#include <cmath>
#include "matrices.hpp"
#include "common.hpp"
```

Include dependency graph for matrices.cpp:

7.77 src/dynamic/matrices.hpp File Reference

```
#include <Eigen/Dense>
#include "common.hpp"
```

Include dependency graph for matrices.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Matrices

7.78 src/main.cpp File Reference

```
#include <iostream>
#include <cxxopts.hpp>
#include "simulation/simulation.hpp"
#include "simulation/uav_state.hpp"
#include "dynamic/forces.hpp"
#include "common.hpp"
#include "params.hpp"
```

Include dependency graph for main.cpp:

Functions

- void parseArgs (int argc, char **argv, UAVparams *params, Params &p)

  Parse CL arguments.
- int main (int argc, char **argv)

7.78.1 Function Documentation

7.78.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

7.78.1.2 parseArgs()

```
void parseArgs (
            int argc,
            char ** argv,
            UAVparams * params,
            Params & p )
```

Parse CL arguments.

Parameters

argc	number of argument
argv	argument array
params	pointer to UAVparams instant that should be filled
p	internal params reference

7.79 src/params.cpp File Reference

```
#include "params.hpp"
#include <iostream>
```

Include dependency graph for params.cpp:

7.80 src/params.hpp File Reference

```
#include <string>
```

Include dependency graph for params.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Params

  Simulation parameters.

7.81 src/simulation/atmosphere.cpp File Reference

```
#include "atmosphere.hpp"
#include <iostream>
```

Include dependency graph for atmosphere.cpp:

7.82 src/simulation/atmosphere.hpp File Reference

```
#include <Eigen/Dense>
#include <atomic>
#include "common.hpp"
```

Include dependency graph for atmosphere.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct AtmosphereInfo

  DTO containing atmosphere information.
- class Atmosphere

  Representation of atmosphere where aircrafts fly.

7.83 src/simulation/control.cpp File Reference

```
#include <Eigen/Dense>
#include <zmq.hpp>
#include <sstream>
#include <iostream>
#include "control.hpp"
#include "uav_state.hpp"
#include "common.hpp"
#include "../dynamic/matrices.hpp"
#include "../aircraft/aircraft.hpp"
#include "../defines.hpp"
#include "atmosphere.hpp"
```

Include dependency graph for control.cpp:

Functions

- void setAtmosphere (std::string &msg_str, zmq::socket_t &sock)
- void setForce (Aircraft *aircraft, std::string &msg_str, zmq::socket_t &sock)
- void setSpeed (Aircraft *aircraft, std::string &msg_str, zmq::socket_t &sock)
- bool control (Aircraft *aircraft, std::string &msg_str, zmq::socket_t &sock)
- void shoot (Aircraft *aircraft, std::string &msg_str, zmq::socket_t &sock)
- void dropCargo (Aircraft *aircraft, std::string &msg_str, zmq::socket_t &sock)
- bool isNormal (double factor)
- void solidSurfColision (Aircraft *aircraft, std::string &msg_str, zmq::socket_t &sock)
- void setControlSurface (Aircraft *aircraft, std::string &msg_str, zmq::socket_t &sock)
- void startJet (Aircraft *aircraft, std::string &msg_str, zmq::socket_t &sock)
- void setHinges (Aircraft *aircraft, std::string &msg_str, zmq::socket_t &sock)
- void controlListenerJob (zmq::context_t *ctx, std::string address, Aircraft *aircraft)

  Job of control listener thread. Listen for new control command and handle them.

7.83.1 Function Documentation

7.83.1.1 control()

```
bool control (
            Aircraft * aircraft,
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.83.1.2 controlListenerJob()

```
void controlListenerJob (
            zmq::context_t * ctx,
            std::string address,
            Aircraft * aircraft )
```

Job of control listener thread. Listen for new control command and handle them.

Parameters

ctx	zero mq context
address	address of listener socket
aircraft	pointer to aircraft

7.83.1.3 dropCargo()

```
void dropCargo (
            Aircraft * aircraft,
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.83.1.4 isNormal()

```
bool isNormal (
            double factor )
```

7.83.1.5 setAtmosphere()

```
void setAtmosphere (
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.83.1.6 setControlSurface()

```
void setControlSurface (
            Aircraft * aircraft,
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.83.1.7 setForce()

```
void setForce (
            Aircraft * aircraft,
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.83.1.8 setHinges()

```
void setHinges (
            Aircraft * aircraft,
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.83.1.9 setSpeed()

```
void setSpeed (
            Aircraft * aircraft,
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.83.1.10 shoot()

```
void shoot (
            Aircraft * aircraft,
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.83.1.11 solidSurfColision()

```
void solidSurfColision (
            Aircraft * aircraft,
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.83.1.12 startJet()

```
void startJet (
            Aircraft * aircraft,
            std::string & msg_str,
            zmq::socket_t & sock )
```

7.84 src/simulation/control.hpp File Reference

```
#include <zmq.hpp>
#include "uav_state.hpp"
#include "../dynamic/matrices.hpp"
#include "../defines.hpp"
#include "../aircraft/aircraft.hpp"
```

Include dependency graph for control.hpp: This graph shows which files directly or indirectly include this file:

Functions

- void controlListenerJob (zmq::context_t *ctx, std::string address, Aircraft *aircraft)

  Job of control listener thread. Listen for new control command and handle them.

7.84.1 Function Documentation

7.84.1.1 controlListenerJob()

```
void controlListenerJob (
            zmq::context_t * ctx,
            std::string address,
            Aircraft * aircraft )
```

Job of control listener thread. Listen for new control command and handle them.

Parameters

ctx	zero mq context
address	address of listener socket
aircraft	pointer to aircraft

7.85 src/simulation/simulation.cpp File Reference

```
#include <Eigen/Dense>
#include <zmq.hpp>
#include <iostream>
```

```
#include <cstdio>
#include <thread>
#include <mutex>
#include <filesystem>
#include <chrono>
#include "simulation.hpp"
#include "uav_state.hpp"
#include "common.hpp"
#include "control.hpp"
#include "../defines.hpp"
#include "../params.hpp"
```

Include dependency graph for simulation.cpp:

7.86 src/simulation/simulation.hpp File Reference

```
#include <zmq.hpp>
#include <memory>
#include "common.hpp"
#include "uav_state.hpp"
#include "../dynamic/forces.hpp"
#include "../dynamic/matrices.hpp"
#include "../aircraft/aircraft.hpp"
#include "atmosphere.hpp"
```

Include dependency graph for simulation.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Simulation

7.87 src/simulation/uav_state.cpp File Reference

```
#include "uav_state.hpp"
#include "../dynamic/matrices.hpp"
#include "../params.hpp"
```

Include dependency graph for uav_state.cpp:

Functions

- std::ostream & operator<< (std::ostream &outs, const UAVstate &state)

7.87.1 Function Documentation

7.87.1.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & outs,
            const UAVstate & state )
```

7.88 src/simulation/uav_state.hpp File Reference

```
#include <Eigen/Dense>
#include <atomic>
#include <condition_variable>
#include <mutex>
#include "common.hpp"
#include "../defines.hpp"
```

Include dependency graph for uav_state.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct UAVstate

7.89 tests/test1.cpp File Reference

```
#include <iostream>
#include <Eigen/Dense>
#include "../src/RK4.hpp"
```

Include dependency graph for test1.cpp:

Functions

- Eigen::Vector2d fun (double t, Eigen::Vector2d y)
- int test1 ()

7.89.1 Function Documentation

7.89.1.1 fun()

```
Eigen::Vector2d fun (
            double t,
            Eigen::Vector2d y )
```

7.89.1.2 test1()

```
int test1 ( )
```

7.90 tests/test2.cpp File Reference

```
#include <iostream>
#include <Eigen/Dense>
#include "constants.hpp"
```

Include dependency graph for test2.cpp:

Functions

- int main ()

7.90.1 Function Documentation

7.90.1.1 main()

```
int main ( )
```

7.91 tests/test3.cpp File Reference

```
#include <iostream>
#include <Eigen/Dense>
#include "constants.hpp"
#include "forces.hpp"
#include "RK4.hpp"
#include "common.hpp"
```

Include dependency graph for test3.cpp:

Functions

- Eigen::Vector4d fun (double t, Eigen::Vector4d om)
- int main ()

7.91.1 Function Documentation

7.91.1.1 fun()

```
Eigen::Vector4d fun (
            double t,
            Eigen::Vector4d om )
```

7.91.1.2 main()

```
int main ( )
```

7.92 tests/test4.cpp File Reference

```
#include <iostream>
#include <Eigen/Dense>
#include "common.hpp"
#include "uav_state.hpp"
```
Include dependency graph for test4.cpp:

Functions

- int main ()

7.92.1 Function Documentation

7.92.1.1 main()

```
int main ( )
```

7.93 tests/test5.cpp File Reference

```
#include <iostream>
#include <thread>
#include <chrono>
#include "timed_loop.hpp"
```
Include dependency graph for test5.cpp:

Functions

- int main ()

7.93.1 Function Documentation

7.93.1.1 main()

```
int main ( )
```

7.94 tests/zmq_recv.py File Reference

Namespaces

- zmq_recv

Variables

- zmq_recv.context = zmq.Context()
- zmq_recv.socket = context.socket(zmq.SUB)
- string zmq_recv.topicfilter = "pos"
- zmq_recv.s = socket.recv_string()

7.95 tests/zmq_recv_last.py File Reference

Namespaces

- zmq_recv_last

Variables

- zmq_recv_last.context = zmq.Context()
- zmq_recv_last.socket = context.socket(zmq.SUB)
- string zmq_recv_last.topicfilter = ""
- zmq_recv_last.s = socket.recv_string()

7.96 tests/zmq_send.py File Reference

Namespaces

- zmq_send

Variables

- zmq_send.context = zmq.Context()
- zmq_send.socket = context.socket(zmq.PUB)
- int zmq_send.counter = 0

7.97 tests/zmq_send_tcp.py File Reference

Namespaces

- zmq_send_tcp

Variables

- zmq_send_tcp.context = zmq.Context()
- zmq_send_tcp.socket = context.socket(zmq.PUB)
- float zmq_send_tcp.angle = 0.0

Index