# UAV drop physic

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 PIFF

## 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 cofficient. Protect again diving objects in horizontal wall*
- const double DEFAULT_AIR_DENSITY = 1.224

  *Dry air density in normal conditions in kg/m3.*

### 5.2.1 Detailed Description

Simulation constants.

### 5.2.2 Variable Documentation

#### 5.2.2.1 DEFAULT_AIR_DENSITY

```
const double def::DEFAULT_AIR_DENSITY = 1.224
```

Dry air density in normal conditions in kg/m3.

#### 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 cofficient. 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.

# 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 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

### 6.3.1 Constructor & Destructor Documentation

#### 6.3.1.1 Ammo() [1/2]

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

#### 6.3.1.2 Ammo() [2/2]

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

### 6.3.2 Member Function Documentation

#### 6.3.2.1 getV0()

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

get start velocity of ammo when launched

**Returns**

start velocity vector

#### 6.3.2.2 operator=()

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

### 6.3.3 Member Data Documentation

#### 6.3.3.1 _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

## 6.4 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

## 6.4.1 Constructor & Destructor Documentation

#### 6.4.1.1 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 |

#### 6.4.1.2 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 |

## 6.4.2 Member Function Documentation

#### 6.4.2.1 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

#### 6.4.2.2 clear()

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

clear internal state

Implements Controller.

#### 6.4.2.3 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

## 6.5 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

### 6.5.1 Constructor & Destructor Documentation

#### 6.5.1.1 Cargo() [1/2]

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

#### 6.5.1.2 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

## 6.6 Controller Class Reference

```
#include <controller.hpp>
```

Inheritance diagram for Controller:

### Public Member Functions

- Controller ()

  *Default constructor.*
- ∼Controller ()

  *Empty deconstructor for derived classes.*
- 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

## 6.6.1 Constructor & Destructor Documentation

### 6.6.1.1 Controller()

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

Default constructor.

### 6.6.1.2 ∼Controller()

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

Empty deconstructor for derived classes.

## 6.6.2 Member Function Documentation

### 6.6.2.1 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

#### 6.6.2.2 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::PIFF, controllers::PID, and controllers::DoubleSetpoint.

#### 6.6.2.3 clear()

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

clear internal state

Implemented in controllers::PIFF, controllers::PID, controllers::DoubleSetpoint, and controllers::BangBang.

#### 6.6.2.4 clone()

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

virtual clone method

Implemented in controllers::PIFF, controllers::PID, controllers::DoubleSetpoint, and controllers::BangBang.

#### 6.6.2.5 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 |

#### 6.6.2.6 set_dt()

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

Set new time step.

**Parameters**

|  |  |
|---|---|
| *dt* | new time step |

### 6.6.3 Member Data Documentation

#### 6.6.3.1 _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

## 6.7 ControllerTest Class Reference

Inheritance diagram for ControllerTest:

Collaboration diagram for ControllerTest:

### Protected Member Functions

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

### 6.7.1 Member Function Documentation

#### 6.7.1.1 SetUp()

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

#### 6.7.1.2 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

## 6.8 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

### 6.8.1 Detailed Description

Aircraft's control surfaces.

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 ControlSurfaces() [1/2]

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

#### 6.8.2.2 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 |

## 6.8.3 Member Function Documentation

### 6.8.3.1 getCoefficients()

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

### 6.8.3.2 getNoOfSurface()

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

### 6.8.3.3 getValues()

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

### 6.8.3.4 restoreTrim()

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

### 6.8.3.5 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

## 6.9 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

### 6.9.1 Constructor & Destructor Documentation

#### 6.9.1.1 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 |

#### 6.9.1.2 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 |

### 6.9.2 Member Function Documentation

#### 6.9.2.1 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.

#### 6.9.2.2 clear()

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

clear internal state

Implements Controller.

#### 6.9.2.3 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

## 6.10 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]

### 6.10.1 Detailed Description

Drive propelling aircraft.

### 6.10.2 Member Data Documentation

#### 6.10.2.1 axis

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

#### 6.10.2.2 hinges

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

#### 6.10.2.3 noOfHinges

```
int Drive::noOfHinges
```

#### 6.10.2.4 position

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

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

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

## 6.11 DropTest Class Reference

Inheritance diagram for DropTest:

Collaboration diagram for DropTest:

### Protected Member Functions

- void SetUp () override
- void TearDown () override
- void sendControlMessage (std::string msg, bool should_response=true, bool should_response_ok=true)
- int recvState (std::string &response_str)
- auto getParsedState ()
- void collectSample (const int N=10)

### 6.11.1 Member Function Documentation

#### 6.11.1.1 collectSample()

```
void DropTest::collectSample (
             const int N = 10 )  [inline], [protected]
```

#### 6.11.1.2 getParsedState()

```
auto DropTest::getParsedState ( )  [inline], [protected]
```

#### 6.11.1.3 recvState()

```
int DropTest::recvState (
             std::string & response_str )  [inline], [protected]
```

#### 6.11.1.4 sendControlMessage()

```
void DropTest::sendControlMessage (
             std::string msg,
             bool should_response = true,
             bool should_response_ok = true )  [inline], [protected]
```

#### 6.11.1.5 SetUp()

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

#### 6.11.1.6 TearDown()

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

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

- tests/integration_test.cpp

## 6.12 EKFScalers Struct Reference

Scalers for EKF.

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

### Public Attributes

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

### 6.12.1 Detailed Description

Scalers for EKF.

### 6.12.2 Member Data Documentation

#### 6.12.2.1 baroScaler

```
double EKFScalers::baroScaler
```

#### 6.12.2.2 predictScaler

```
double EKFScalers::predictScaler
```

#### 6.12.2.3 updateScaler

```
double EKFScalers::updateScaler
```

#### 6.12.2.4 zScaler

```
double EKFScalers::zScaler
```

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

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

## 6.13 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.*

### 6.13.1 Detailed Description

Hinge connecting aircraft with drives.

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 Hinge() [1/3]

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

#### 6.13.2.2 Hinge() [2/3]

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

#### 6.13.2.3 Hinge() [3/3]

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

### 6.13.3 Member Function Documentation

#### 6.13.3.1 getRot()

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

Get rotattion matrix of orientation change due to hinge.

**Returns**

rotation matrix

#### 6.13.3.2 operator=()

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

#### 6.13.3.3 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

## 6.14 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

### 6.14.1 Detailed Description

Jet rocket engine.

### 6.14.2 Member Function Documentation

#### 6.14.2.1 getLastThrust()

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

get last calculated thrust

**Returns**

last calculated thrust

#### 6.14.2.2 getThrust()

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

get thrust in specific time

**Parameters**

| | |
|---|---|
| *time* | timestamp |

**Returns**

thrust value in Newtons

#### 6.14.2.3 start()

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

start jet engine

**Parameters**

| | |
|---|---|
| *time* | timestamp of start |

**Returns**

true if start succesful, false if already started

### 6.14.3 Member Data Documentation

#### 6.14.3.1 phases

```
int Jet::phases
```

#### 6.14.3.2 thrust

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

#### 6.14.3.3 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

## 6.15 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 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)

### 6.15.1 Detailed Description

Load of aircraft that can be droped or launched.

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 Load() [1/2]

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

#### 6.15.2.2 Load() [2/2]

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

### 6.15.3 Member Function Documentation

#### 6.15.3.1 getMass()

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

get mass of load

**Returns**

mass

#### 6.15.3.2 getOffset()

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

get offset of load

**Returns**

offset vector

#### 6.15.3.3 operator=()

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

#### 6.15.3.4 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

## 6.16 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.*

### 6.16.1 Detailed Description

Log vector data with timestamp in file.

### 6.16.2 Constructor & Destructor Documentation

#### 6.16.2.1 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 |

#### 6.16.2.2 ∼Logger()

```
Logger::∼Logger ( )
```

Deconstructor.

### 6.16.3 Member Function Documentation

#### 6.16.3.1 log() [1/2]

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

Log one row.

**Parameters**

| | |
|---|---|
| *time* | timestamp |
| *args* | list of doubles |

#### 6.16.3.2 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 |

#### 6.16.3.3 setFmt()

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

Set new format if was not known in constructor.

**Parameters**

| | |
|---|---|
| *fmt* | new format |

#### 6.16.3.4 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

## 6.17 ObjParams Class Reference

Single obj parameters.

```
#include <state.hpp>
```

### Public Member Functions

- ObjParams (double mass, double CS_coff)

  *Constructor.*
- ObjParams (ObjParams &&rhs)

  *Moving constructor.*
- void setWind (Eigen::Vector3d newWind)

  *Set wind vector affecting on object.*
- Eigen::Vector3d getWind ()

  *Get wind vector.*
- void setForce (Eigen::Vector3d newForce)

  *Set outer force applied to object.*
- Eigen::Vector3d getForce ()

  *Get outer force.*

## Public Attributes

- const int id

    *object id*
- const double mass

    *object mass*
- const double CS_coff

    *aerodynamic drag force cofficent multipled by aerodynamic field*

### 6.17.1 Detailed Description

Single obj parameters.

### 6.17.2 Constructor & Destructor Documentation

#### 6.17.2.1 ObjParams() [1/2]

```
ObjParams::ObjParams (
            double mass,
            double CS_coff )  [inline]
```

Constructor.

**Parameters**

| | |
|---|---|
| *mass* | object mass |
| *CS_coff* | aerodynamic drag force cofficent multipled by aerodynamic field |

#### 6.17.2.2 ObjParams() [2/2]

```
ObjParams::ObjParams (
            ObjParams && rhs )  [inline]
```

Moving constructor.

**Parameters**

| | |
|---|---|
| *rhs* | other instant that should be consumed |

### 6.17.3 Member Function Documentation

#### 6.17.3.1 getForce()

```
Eigen::Vector3d ObjParams::getForce ( )
```

Get outer force.

**Returns**

outer force vector in N

#### 6.17.3.2 getWind()

```
Eigen::Vector3d ObjParams::getWind ( )
```

Get wind vector.

**Returns**

wind speed vector in m/s

#### 6.17.3.3 setForce()

```
void ObjParams::setForce (
             Eigen::Vector3d newForce )
```

Set outer force applied to object.

**Parameters**

| | |
|---|---|
| *newForce* | new force vector in N |

#### 6.17.3.4 setWind()

```
void ObjParams::setWind (
             Eigen::Vector3d newWind )
```

Set wind vector affecting on object.

**Parameters**

| | |
|---|---|
| *newWind* | new wind speed vector in m/s |

### 6.17.4 Member Data Documentation

#### 6.17.4.1 CS_coff

```
const double ObjParams::CS_coff
```

aerodynamic drag force cofficent multipled by aerodynamic field

#### 6.17.4.2 id

```
const int ObjParams::id
```

object id

#### 6.17.4.3 mass

```
const double ObjParams::mass
```

object mass

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

- src/state.hpp
- src/state.cpp

## 6.18 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.*

## 6.18.1 Detailed Description

Ordinal differencial equation solver.

## 6.18.2 Member Enumeration Documentation

### 6.18.2.1 ODEMethod

```
enum ODE::ODEMethod
```

Supported solving method.

**Enumerator**

| | |
|---|---|
| Euler | |
| Heun | |
| RK4 | |
| PC2 | |
| PC4 | |
| NONE | |

## 6.18.3 Constructor & Destructor Documentation

### 6.18.3.1 ODE()

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

Constructor.

### 6.18.3.2 ∼ODE()

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

Virtual deconstructor.

## 6.18.4 Member Function Documentation

### 6.18.4.1 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

### 6.18.4.2 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

#### 6.18.4.3 getMicrosteps() [1/2]

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

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

**Returns**

microsteps

#### 6.18.4.4 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

#### 6.18.4.5 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

## 6.19 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

### 6.19.1 Detailed Description

Explicit Euler algorithm.

### 6.19.2 Constructor & Destructor Documentation

#### 6.19.2.1 ODE_Euler()

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

### 6.19.3 Member Function Documentation

#### 6.19.3.1 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

## 6.20 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::VectorXd)> rhs_fun, double h) override

  *One step of explicit solving algorithm.*

## Additional Inherited Members

## 6.20.1 Detailed Description

Second order explicit Heun algorithm.

## 6.20.2 Constructor & Destructor Documentation

### 6.20.2.1 ODE_Heun()

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

## 6.20.3 Member Function Documentation

### 6.20.3.1 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

## 6.21 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

### 6.21.1 Detailed Description

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

### 6.21.2 Constructor & Destructor Documentation

#### 6.21.2.1 ODE_PC2()

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

### 6.21.3 Member Function Documentation

#### 6.21.3.1 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

## 6.22 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

### 6.22.1 Detailed Description

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

### 6.22.2 Constructor & Destructor Documentation

#### 6.22.2.1 ODE_PC4()

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

### 6.22.3 Member Function Documentation

#### 6.22.3.1 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

## 6.23 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

## 6.23.1 Detailed Description

Fourth order Runge Kutta algorithm.

## 6.23.2 Constructor & Destructor Documentation

### 6.23.2.1 ODE_RK4()

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

## 6.23.3 Member Function Documentation

### 6.23.3.1 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

## 6.24 ODETest Class Reference

Inheritance diagram for ODETest:

Collaboration diagram for ODETest:

### Protected Member Functions

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

### 6.24.1 Member Function Documentation

#### 6.24.1.1 SetUp()

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

#### 6.24.1.2 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

## 6.25 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.*

### 6.25.1 Detailed Description

Simulation parameters.

### 6.25.2 Constructor & Destructor Documentation

#### 6.25.2.1 Params() [1/3]

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

Constructor.

#### 6.25.2.2 Params() [2/3]

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

#### 6.25.2.3 Params() [3/3]

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

#### 6.25.2.4 ∼Params()

```
Params::∼Params ( )
```

Deconstructor.

### 6.25.3 Member Function Documentation

#### 6.25.3.1 getSingleton()

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

Get singleton of Params.

**Returns**

const pointer to Params instance. Return nullptr if not initialized

#### 6.25.3.2 operator=()

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

### 6.25.4 Member Data Documentation

#### 6.25.4.1 ODE_METHOD

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

ODE solving method used in simulation.

#### 6.25.4.2 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

## 6.26 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 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

### 6.26.1 Member Enumeration Documentation

#### 6.26.1.1 AntiWindUpMode

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

Methods of handling windup in controller.

**Enumerator**

| | |
|---|---|
| NONE | |
| CLAMPING | |

## 6.26.2 Constructor & Destructor Documentation

### 6.26.2.1 PID() [1/2]

```
PID::PID (
            double Kp,
            double Ki,
            double Kd,
            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 |
| *min* | saturation - lower range limit |
| *max* | saturation - upper range limit |
| *antiWindUp* | antiwindup method |

### 6.26.2.2 PID() [2/2]

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

Construct controller with parameters from xml.

**Parameters**

| | |
|---|---|
| *controller_node* | xml node with controller params |

## 6.26.3 Member Function Documentation

#### 6.26.3.1 calc()

```
double 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.

#### 6.26.3.2 clear()

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

clear internal state

Implements Controller.

#### 6.26.3.3 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

## 6.27 controllers::PIFF Class Reference

```
#include <PIFF.hpp>
```

Inheritance diagram for controllers::PIFF:

Collaboration diagram for controllers::PIFF:

## Public Types

- enum class AntiWindUpMode { NONE , CLAMPING }

    *Methods of handling windup in controller.*

## Public Member Functions

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

    *Constructor with all PID controller parameters.*
- PIFF (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

## 6.27.1 Member Enumeration Documentation

### 6.27.1.1 AntiWindUpMode

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

Methods of handling windup in controller.

**Enumerator**

| | |
|---|---|
| NONE | |
| CLAMPING | |

## 6.27.2 Constructor & Destructor Documentation

### 6.27.2.1 PIFF() [1/2]

```
PIFF::PIFF (
            double Kp,
            double Ki,
            double Kff,
            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 |
| *Kff* | FF term |
| *min* | saturation - lower range limit |
| *max* | saturation - upper range limit |
| *antiWindUp* | antiwindup method |

#### 6.27.2.2 PIFF() [2/2]

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

Construct controller with parameters from xml.

**Parameters**

| | |
|---|---|
| *controller_node* | xml node with controller params |

### 6.27.3 Member Function Documentation

#### 6.27.3.1 calc()

```
double PIFF::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.

#### 6.27.3.2 clear()

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

clear internal state

Implements Controller.

#### 6.27.3.3 clone()

```
std::unique_ptr< Controller > PIFF::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/PIFF.hpp
- lib/UAV_common/src/controllers/impl/PIFF.cpp

## 6.28 Projectile Struct Reference

```
#include <projectile_parser.hpp>
```

### Public Attributes

- int id
- Eigen::Vector3d position
- Eigen::Vector3d velocity

### 6.28.1 Member Data Documentation

#### 6.28.1.1 id

```
int Projectile::id
```

#### 6.28.1.2 position

```
Eigen::Vector3d Projectile::position
```

#### 6.28.1.3 velocity

```
Eigen::Vector3d Projectile::velocity
```

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

- tests/projectile_parser.hpp

## 6.29 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

### 6.29.1 Detailed Description

Rotor engine with controlled speed.

### 6.29.2 Member Data Documentation

#### 6.29.2.1 direction

```
int Rotor::direction
```

#### 6.29.2.2 forceCoff

```
double Rotor::forceCoff
```

#### 6.29.2.3 hoverSpeed

```
double Rotor::hoverSpeed
```

#### 6.29.2.4 maxSpeed

```
double Rotor::maxSpeed
```

#### 6.29.2.5 timeConstant

```
double Rotor::timeConstant
```

#### 6.29.2.6 torqueCoff

```
double Rotor::torqueCoff
```

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

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

## 6.30 SensorParams Struct Reference

Base parameters of a sensor.

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

## Public Attributes

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

### 6.30.1 Detailed Description

Base parameters of a sensor.

### 6.30.2 Member Data Documentation

#### 6.30.2.1 bias

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

#### 6.30.2.2 name

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

#### 6.30.2.3 refreshTime

```
double SensorParams::refreshTime
```

#### 6.30.2.4 sd

```
double SensorParams::sd
```

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

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

## 6.31 Simulation Class Reference

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

## Public Member Functions

- Simulation (const Params &params)

  *Constructor.*
- ∼Simulation ()

  *Deconstructor.*
- void run ()

  *Run simulation.*
- int addObj (double mass, double CS, Eigen::Vector3d pos, Eigen::Vector3d vel=Eigen::Vector3d())

  *Add new object to simulation.*
- void removeObj (int id)

  *Remove object from simulation.*
- void addCommand (std::string msg, zmq::socket_t &sock)

  *Handle add new object command.*
- void updateWind (std::string msg, zmq::socket_t &sock)

  *Handle update wind command.*
- void updateForce (std::string msg, zmq::socket_t &sock)

  *Handle update force command.*
- void solidSurfColision (std::string &msg_str, zmq::socket_t &sock)

  *Handle solid surface collision command.*
- void calcImpulseForce (int id, double COR, double mi_static, double mi_dynamic, Eigen::Vector3d surface←↩ Normal)

  *Calculates object state after collision with given surface.*

## 6.31.1 Constructor & Destructor Documentation

### 6.31.1.1 Simulation()

```
Simulation::Simulation (
            const Params & params )
```

Constructor.

**Parameters**

| | |
|---|---|
| *params* | simulation params |

### 6.31.1.2 ∼Simulation()

```
Simulation::∼Simulation ( )
```

Deconstructor.

## 6.31.2 Member Function Documentation

#### 6.31.2.1 addCommand()

```
void Simulation::addCommand (
             std::string msg,
             zmq::socket_t & sock )
```

Handle add new object command.

**Parameters**

| | |
|---|---|
| *msg* | message content |
| *sock* | zmq socket reply is send by |

#### 6.31.2.2 addObj()

```
int Simulation::addObj (
             double mass,
             double CS,
             Eigen::Vector3d pos,
             Eigen::Vector3d vel = Eigen::Vector3d() )
```

Add new object to simulation.

**Parameters**

| | |
|---|---|
| *mass* | obj mass |
| *CS* | aerodynamic drag force cofficent multipled by aerodynamic field |
| *pos* | start position of object |
| *vel* | start velocity of object |

**Returns**

id of added object

#### 6.31.2.3 calcImpulseForce()

```
void Simulation::calcImpulseForce (
             int id,
             double COR,
             double mi_static,
             double mi_dynamic,
             Eigen::Vector3d surfaceNormal )
```

Calculates object state after collision with given surface.

**Parameters**

| | |
|---|---|
| *id* | object id |
| *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 |
| *surfaceNormal* | surface normal vector |

#### 6.31.2.4 removeObj()

```
void Simulation::removeObj (
            int id )
```

Remove object from simulation.

**Parameters**

| | |
|---|---|
| *id* | object id |

#### 6.31.2.5 run()

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

Run simulation.

#### 6.31.2.6 solidSurfColision()

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

Handle solid surface collision command.

**Parameters**

| | |
|---|---|
| *msg* | message content |
| *sock* | zmq socket reply is send by |

#### 6.31.2.7 updateForce()

```
void Simulation::updateForce (
            std::string msg,
            zmq::socket_t & sock )
```

Handle update force command.

**Parameters**

| | |
|---|---|
| *msg* | message content |
| *sock* | zmq socket reply is send by |

#### 6.31.2.8 updateWind()

```
void Simulation::updateWind (
            std::string msg,
            zmq::socket_t & sock )
```

Handle update wind command.

**Parameters**

| | |
|---|---|
| *msg* | message content |
| *sock* | zmq socket reply is send by |

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

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

## 6.32 State Class Reference

```
#include <state.hpp>
```

### Public Member Functions

- State ()

  *Constructor.*
- Eigen::VectorXd getState ()

  *Get full state as vector.*
- void updateState (Eigen::VectorXd newState)

  *Update state.*
- void updateWind (int id, Eigen::Vector3d newWind)

  *update wind speed for obj specified by id*

- void updateForce (int id, Eigen::Vector3d newForce)

    *update outer force applied to object specified by id*
- int addObj (double mass, double CS_coff, Eigen::Vector3d pos, Eigen::Vector3d vel=Eigen::Vector3d())

    *Add new object to simulation.*
- void removeObj (int id)

    *remove object specified by id*
- std::string to_string ()

    *Serialize state to string.*
- int findIndex (int id)

    *Find index of object specified by id.*
- int getNoObj ()

    *Get number of active object in simulation.*
- ObjParams ∗ getParams (int index)

    *get params of object specified by index*
- Eigen::Vector3d getPos (int index)

    *Get position of object specified by index.*
- Eigen::Vector3d getVel (int index)

    *Get velocity of object specified by index.*
- void setVel (int index, Eigen::Vector3d newVel)

    *Override velocity of object, for example after collision.*

## Public Attributes

- std::mutex stateMutex

    *mutex to manipule on state responses*
- double real_time

    *time of simulation*
- Status status

    *status for timed loop*

### 6.32.1 Constructor & Destructor Documentation

#### 6.32.1.1 State()

```
State::State ( )
```

Constructor.

### 6.32.2 Member Function Documentation

#### 6.32.2.1 addObj()

```
int State::addObj (
              double mass,
              double CS_coff,
              Eigen::Vector3d pos,
              Eigen::Vector3d vel = Eigen::Vector3d() )
```

Add new object to simulation.

**Parameters**

| | |
|---|---|
| *mass* | mass of object |
| *CS_coff* | aerodynamic drag force cofficent multipled by aerodynamic field |
| *pos* | start position |
| *vel* | start velocity |

**Returns**

id of added object

#### 6.32.2.2 findIndex()

```
int State::findIndex (
             int id )
```

Find index of object specified by id.

**Parameters**

| | |
|---|---|
| *id* | object id |

**Returns**

object index

#### 6.32.2.3 getNoObj()

```
int State::getNoObj ( )  [inline]
```

Get number of active object in simulation.

**Returns**

number of object

#### 6.32.2.4 getParams()

```
ObjParams* State::getParams (
             int index )  [inline]
```

get params of object specified by index

**Parameters**

| | |
|---|---|
| *index* | index of object |

**Returns**

pointer to object params

### 6.32.2.5 getPos()

```
Eigen::Vector3d State::getPos (
             int index )  [inline]
```

Get position of object specified by index.

**Parameters**

| | |
|---|---|
| *index* | index of object |

**Returns**

position vector

### 6.32.2.6 getState()

```
Eigen::VectorXd State::getState ( )
```

Get full state as vector.

**Returns**

state vector

### 6.32.2.7 getVel()

```
Eigen::Vector3d State::getVel (
             int index )  [inline]
```

Get velocity of object specified by index.

**Parameters**

|  |  |
|---|---|
| *index* | index of object |

**Returns**

velocity of object

#### 6.32.2.8 removeObj()

```
void State::removeObj (
             int id )
```

remove object specified by id

**Parameters**

|  |  |
|---|---|
| *id* | id of removing object |

#### 6.32.2.9 setVel()

```
void State::setVel (
             int index,
             Eigen::Vector3d newVel )  [inline]
```

Override velocity of object, for example after collision.

**Parameters**

|  |  |
|---|---|
| *index* | index of object |
| *newVel* | new velocity vector |

#### 6.32.2.10 to_string()

```
std::string State::to_string ( )
```

Serialize state to string.

**Returns**

serialized state

#### 6.32.2.11 updateForce()

```
void State::updateForce (
             int id,
             Eigen::Vector3d newForce )
```

update outer force applied to object specified by id

**Parameters**

| | |
|---|---|
| *id* | id of updated obj |
| *newForce* | new force value |

#### 6.32.2.12 updateState()

```
void State::updateState (
             Eigen::VectorXd newState )
```

Update state.

**Parameters**

| | |
|---|---|
| *newState* | new state vector |

#### 6.32.2.13 updateWind()

```
void State::updateWind (
             int id,
             Eigen::Vector3d newWind )
```

update wind speed for obj specified by id

**Parameters**

| | |
|---|---|
| *id* | id of updated obj |
| *newWind* | new wind speed vector |

### 6.32.3 Member Data Documentation

#### 6.32.3.1 real_time

```
double State::real_time
```

time of simulation

#### 6.32.3.2 stateMutex

```
std::mutex State::stateMutex
```

mutex to manipule on state responses

#### 6.32.3.3 status

```
Status State::status
```

status for timed loop

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

- src/state.hpp
- src/state.cpp

## 6.33 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*

### 6.33.1 Detailed Description

Simulation of real-time synchronized loop.

### 6.33.2 Constructor & Destructor Documentation

#### 6.33.2.1 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 |

### 6.33.3 Member Function Documentation

#### 6.33.3.1 go() [1/2]

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

start infinite loop

#### 6.33.3.2 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

## 6.34 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

## 6.34.1 Detailed Description

Parsed UAV configuration from XML.

### 6.34.2 Constructor & Destructor Documentation

#### 6.34.2.1 UAVparams()

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

Initialize default data.

#### 6.34.2.2 ∼UAVparams()

```
UAVparams::∼UAVparams ( )
```

### 6.34.3 Member Function Documentation

#### 6.34.3.1 getRotorHoverSpeeds()

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

#### 6.34.3.2 getRotorMaxSpeeds()

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

#### 6.34.3.3 getRotorTimeContants()

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

#### 6.34.3.4 getSingleton()

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

#### 6.34.3.5 loadConfig()

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

### 6.34.4 Member Data Documentation

#### 6.34.4.1 aero_coffs

```
AeroCoefficients UAVparams::aero_coffs
```

#### 6.34.4.2 ahrs

```
AHRSParams UAVparams::ahrs
```

#### 6.34.4.3 ammo

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

#### 6.34.4.4 cargo

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

#### 6.34.4.5 controllers

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

#### 6.34.4.6 ekf

```
EKFScalers UAVparams::ekf
```

#### 6.34.4.7 initialMode

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

#### 6.34.4.8 initialOrientation

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

#### 6.34.4.9 initialPosition

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

#### 6.34.4.10 initialVelocity

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

#### 6.34.4.11 instantRun

```
bool UAVparams::instantRun
```

#### 6.34.4.12 Ix

```
double UAVparams::Ix
```

#### 6.34.4.13 Ixy

```
double UAVparams::Ixy
```

#### 6.34.4.14 Ixz

```
double UAVparams::Ixz
```

#### 6.34.4.15 Iy

```
double UAVparams::Iy
```

#### 6.34.4.16 Iyz

```
double UAVparams::Iyz
```

#### 6.34.4.17 Iz

```
double UAVparams::Iz
```

#### 6.34.4.18 jets

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

#### 6.34.4.19 m

```
double UAVparams::m
```

#### 6.34.4.20 name

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

#### 6.34.4.21 noOfAmmo

```
int UAVparams::noOfAmmo
```

#### 6.34.4.22 noOfCargo

```
int UAVparams::noOfCargo
```

#### 6.34.4.23 noOfJets

```
int UAVparams::noOfJets
```

#### 6.34.4.24 noOfRotors

```
int UAVparams::noOfRotors
```

#### 6.34.4.25 rotorMixer

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

#### 6.34.4.26 rotors

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

#### 6.34.4.27 sensors

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

#### 6.34.4.28 surfaceMixer

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

#### 6.34.4.29 surfaces

```
ControlSurfaces UAVparams::surfaces
```

#### 6.34.4.30 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

# 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 "]"
```

## 7.3 build/CMakeFiles/drop.dir/src/main.cpp.o.d File Reference

## 7.4 build/CMakeFiles/drop.dir/src/params.cpp.o.d File Reference

## 7.5 build/CMakeFiles/drop.dir/src/simulation.cpp.o.d File Reference

## 7.6 build/CMakeFiles/drop.dir/src/state.cpp.o.d File Reference

## 7.7 build/CMakeFiles/integration_test.dir/tests/integration_test.cpp.o.d File Reference

## 7.8 build/lib/UAV_common/CMake↩ Files/common.dir/src/components/control_surfaces.cpp.o.d File Reference

## 7.9 build/lib/UAV_common/CMake↩ Files/common.dir/src/components/drive.cpp.o.d File Reference

## 7.10 build/lib/UAV_common/CMake↩ Files/common.dir/src/components/hinge.cpp.o.d File Reference

## 7.11 build/lib/UAV_common/CMake↩ Files/common.dir/src/components/loads.cpp.o.d File Reference

## 7.12 build/lib/UAV_common/CMake↩ Files/common.dir/src/controllers/controller.cpp.o.d File Reference

## 7.13 build/lib/UAV_common/CMake↩ Files/common.dir/src/controllers/impl/bang_bang.cpp.o.d File Reference

## 7.14 build/lib/UAV_common/CMake↩ Files/common.dir/src/controllers/impl/double_setpoint.cpp.o.d File Reference

## 7.15 build/lib/UAV_common/CMake↩ Files/common.dir/src/controllers/impl/PID.cpp.o.d File Reference

## 7.16 build/lib/UAV_common/CMakeFiles/common.dir/src/PID/PID.cpp.o.d File Reference

## 7.17 build/lib/UAV_common/CMake↩ Files/common.dir/src/controllers/impl/PIFF.cpp.o.d File Reference

## 7.18 build/lib/UAV_common/CMake↩ Files/common.dir/src/logger/logger.cpp.o.d File Reference

```
#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:

## 7.26 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

### 7.26.1 Function Documentation

#### 7.26.1.1 legend()

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

#### 7.26.1.2 plot()

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

#### 7.26.1.3 title()

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

#### 7.26.1.4 xlabel()

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

#### 7.26.1.5 ylabel()

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

### 7.26.2 Variable Documentation

#### 7.26.2.1 clc

```
clc
```

#### 7.26.2.2 csvFiles

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

#### 7.26.2.3 data

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

#### 7.26.2.4 figure

```
figure
```

#### 7.26.2.5 folderPath

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

#### 7.26.2.6 i

```
for i
```

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

#### 7.26.2.7 off

```
hold off
```

#### 7.26.2.8 on

```
hold on
```

#### 7.26.2.9 x

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

#### 7.26.2.10 y

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

## 7.27 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.*

## 7.28 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:

## 7.29 lib/UAV_common/src/components/control_surfaces.cpp File Reference

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

## 7.30 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.*

## 7.31 lib/UAV_common/src/components/drive.cpp File Reference

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

## 7.32 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.*

## 7.33 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)

### 7.33.1 Function Documentation

#### 7.33.1.1 asSkewMatrix()

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

## 7.34 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.*

## 7.35 lib/UAV_common/src/components/loads.cpp File Reference

```
#include "loads.hpp"
#include <limits>
```
Include dependency graph for loads.cpp:

## 7.36 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

## 7.37 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.*

## 7.38 lib/UAV_common/src/controllers/controller.cpp File Reference

```
#include "controller.hpp"
#include "impl/PID.hpp"
#include "impl/PIFF.hpp"
#include "impl/bang_bang.hpp"
#include "impl/double_setpoint.hpp"
#include <cstring>
#include <stdexcept>
```
Include dependency graph for controller.cpp:

## 7.39 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

## 7.40 lib/UAV_common/src/controllers/controller_test.cpp File Reference

```
#include "impl/PID.hpp"
#include "impl/PIFF.hpp"
#include "impl/bang_bang.hpp"
#include "impl/double_setpoint.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"

### 7.40.1 Function Documentation

#### 7.40.1.1 getMethodsToTest()

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

#### 7.40.1.2 INSTANTIATE_TEST_SUITE_P()

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

#### 7.40.1.3 main()

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

#### 7.40.1.4 TEST_P() [1/2]

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

#### 7.40.1.5 TEST_P() [2/2]

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

### 7.40.2 Variable Documentation

#### 7.40.2.1 plot

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

#### 7.40.2.2 plot_directory_name

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

## 7.41 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:

## 7.42 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

## 7.43 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:

## 7.44 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

## 7.45 lib/UAV_common/src/controllers/impl/PID.cpp File Reference

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

## 7.46 lib/UAV_common/src/controllers/impl/PID.hpp File Reference

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

### Classes

- class controllers::PID

### Namespaces

- controllers

## 7.47 lib/UAV_common/src/controllers/impl/PIFF.cpp File Reference

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

## 7.48 lib/UAV_common/src/controllers/impl/PIFF.hpp File Reference

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

### Classes

- class controllers::PIFF

### Namespaces

- controllers

## 7.49 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)

### 7.49.1 Function Documentation

#### 7.49.1.1 shouldLog()

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

## 7.50 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

## 7.50.1 Macro Definition Documentation

#### 7.50.1.1 LOGGER_MASK

```
#define LOGGER_MASK -1
```

## 7.51 lib/UAV_common/src/ode/ode.cpp File Reference

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

## 7.52 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.*

## 7.53 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.*

## 7.54 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)

### 7.54.1 Function Documentation

#### 7.54.1.1 getMethodsToTest()

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

#### 7.54.1.2 INSTANTIATE_TEST_SUITE_P()

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

#### 7.54.1.3 main()

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

#### 7.54.1.4 TEST_F() [1/2]

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

#### 7.54.1.5 TEST_F() [2/2]

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

#### 7.54.1.6 TEST_P() [1/4]

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

#### 7.54.1.7 TEST_P() [2/4]

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

#### 7.54.1.8 TEST_P() [3/4]

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

#### 7.54.1.9 TEST_P() [4/4]

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

## 7.55 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.*

### 7.55.1 Function Documentation

#### 7.55.1.1 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

#### 7.55.1.2 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

## 7.56 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.*

### 7.56.1 Function Documentation

#### 7.56.1.1 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

#### 7.56.1.2 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

## 7.57 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)

### 7.57.1 Function Documentation

#### 7.57.1.1 parseHinge()

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

## 7.58 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.*

## 7.59 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*

### 7.59.1 Enumeration Type Documentation

#### 7.59.1.1 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 |

## 7.60 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:

## 7.61 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.*

## 7.62 src/defines.hpp File Reference

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

### Namespaces

- def

  *Simulation constants.*

### 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 cofficient. Protect again diving objects in horizontal wall*
- const double def::DEFAULT_AIR_DENSITY = 1.224

  *Dry air density in normal conditions in kg/m3.*

## 7.63 src/main.cpp File Reference

```
#include <iostream>
#include <Eigen/Dense>
#include <cxxopts.hpp>
#include "simulation.hpp"
#include "common.hpp"
#include "params.hpp"
```
Include dependency graph for main.cpp:

### Functions

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

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

### 7.63.1 Function Documentation

#### 7.63.1.1 main()

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

#### 7.63.1.2 parseArgs()

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

Parse CL arguments.

**Parameters**

| | |
|---|---|
| *argc* | number of argument |
| *argv* | argument array |
| *p* | reference to params instant that should be filled |

## 7.64 src/params.cpp File Reference

```
#include "params.hpp"
```

```
#include <iostream>
```
Include dependency graph for params.cpp:

## 7.65 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.*

## 7.66 src/simulation.cpp File Reference

```
#include <Eigen/Dense>
#include <zmq.hpp>
#include <iostream>
#include <cstdio>
#include <thread>
#include <mutex>
#include <functional>
#include <map>
#include <filesystem>
#include "simulation.hpp"
#include "common.hpp"
#include "state.hpp"
```
Include dependency graph for simulation.cpp:

### Functions

- bool isNormal (double factor)

### 7.66.1 Function Documentation

#### 7.66.1.1 isNormal()

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

## 7.67 src/simulation.hpp File Reference

```
#include <zmq.hpp>
#include <thread>
#include "state.hpp"
#include <Eigen/Dense>
#include <functional>
#include "common.hpp"
#include "defines.hpp"
#include "params.hpp"
```
Include dependency graph for simulation.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class Simulation

## 7.68 src/state.cpp File Reference

```
#include <Eigen/Dense>
#include <mutex>
#include <iostream>
#include "state.hpp"
#include "common.hpp"
#include "params.hpp"
#include "defines.hpp"
```
Include dependency graph for state.cpp:

## 7.69 src/state.hpp File Reference

```
#include <Eigen/Dense>
#include <zmq.hpp>
#include <thread>
#include <vector>
#include <mutex>
#include <atomic>
#include "common.hpp"
```
Include dependency graph for state.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class ObjParams

  *Single obj parameters.*
- class State

## 7.70 tests/integration_test.cpp File Reference

```
#include <gtest/gtest.h>
#include <zmq.hpp>
#include <Eigen/Dense>
#include <cstdlib>
#include <string>
#include <iostream>
#include <sstream>
#include <thread>
#include <future>
#include <cmath>
#include <chrono>
#include "projectile_parser.hpp"
```

Include dependency graph for integration_test.cpp:

### Classes

- class DropTest

### Functions

- TEST (IntegrationTest, ProgramShowsHelp)

  *Test if program show correct usage.*
- TEST_F (DropTest, ProgramRunsAndExitsCorrectly)

  *Test if program run and exit on command correctly.*
- TEST_F (DropTest, ProgramRunsAndSendState)

  *Test if program run send valid state.*
- TEST_F (DropTest, AddAndRemoveProjectiles)

  *Test if program handle add object and remove object command correctly.*
- TEST_F (DropTest, CheckFreeFall)

  *Test if program simulates free fall correctly.*
- TEST_F (DropTest, LikeParachute)

  *Test if program simulates air drag correctly.*
- TEST_F (DropTest, ForceInfluence)

  *Test if program simulates outer forces influence correctly.*
- TEST_F (DropTest, SolidSurfaceCollision)

  *Test if program simulates solid surface collision correctly.*
- TEST_F (DropTest, StrongWindInfluence)

  *Test if program simulates strong wind influence correctly.*
- int main (int argc, char **argv)

### Variables

- const std::string programPath = "./drop"
- const std::string communicationFolder = "ipc:///tmp/drop_shot"
- zmq::context_t ctx

## 7.70.1 Function Documentation

### 7.70.1.1 main()

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

### 7.70.1.2 TEST()

```
TEST (
            IntegrationTest ,
            ProgramShowsHelp  )
```

Test if program show correct usage.

### 7.70.1.3 TEST_F() [1/8]

```
TEST_F (
            DropTest ,
            AddAndRemoveProjectiles  )
```

Test if program handle add object and remove object command correctly.

### 7.70.1.4 TEST_F() [2/8]

```
TEST_F (
            DropTest ,
            CheckFreeFall  )
```

Test if program simulates free fall correctly.

### 7.70.1.5 TEST_F() [3/8]

```
TEST_F (
            DropTest ,
            ForceInfluence  )
```

Test if program simulates outer forces influence correctly.

### 7.70.1.6 TEST_F() [4/8]

```
TEST_F (
            DropTest ,
            LikeParachute  )
```

Test if program simulates air drag correctly.

### 7.70.1.7 TEST_F() [5/8]

```
TEST_F (
            DropTest ,
            ProgramRunsAndExitsCorrectly  )
```

Test if program run and exit on command correctly.

### 7.70.1.8 TEST_F() [6/8]

```
TEST_F (
            DropTest ,
            ProgramRunsAndSendState  )
```

Test if program run send valid state.

### 7.70.1.9 TEST_F() [7/8]

```
TEST_F (
            DropTest ,
            SolidSurfaceCollision  )
```

Test if program simulates solid surface collision correctly.

### 7.70.1.10 TEST_F() [8/8]

```
TEST_F (
            DropTest ,
            StrongWindInfluence  )
```

Test if program simulates strong wind influence correctly.

### 7.70.2 Variable Documentation

#### 7.70.2.1 communicationFolder

```
const std::string communicationFolder = "ipc:///tmp/drop_shot"
```

#### 7.70.2.2 ctx

```
zmq::context_t ctx
```

#### 7.70.2.3 programPath

```
const std::string programPath = "./drop"
```

## 7.71 tests/projectile_parser.hpp File Reference

```
#include <iostream>
#include <sstream>
#include <vector>
#include <Eigen/Dense>
```

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

### Classes

- struct Projectile

### Functions

- bool checkInputErrors (std::stringstream &ss, char expectedDelimiter, const std::string &errorMessage)
- std::pair< double, std::vector< Projectile > > parseInput (const std::string &input)

### 7.71.1 Function Documentation

#### 7.71.1.1 checkInputErrors()

```
bool checkInputErrors (
            std::stringstream & ss,
            char expectedDelimiter,
            const std::string & errorMessage )
```

#### 7.71.1.2 parseInput()

```
std::pair<double, std::vector<Projectile> > parseInput (
            const std::string & input )
```

# Index