SuperSensorLearningTux

Generated by Doxygen 1.6.1

Mon Feb 1 00:40:51 2010

Contents

Chapter 1

Class Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Class Documentation

3.1 ClassificationTux Class Reference

`#include <ClassificationTux.h>`Inheritance diagram for ClassificationTux::

Public Member Functions

- ClassificationTux ()
- virtual ~ClassificationTux ()
- void threadMethod ()

3.1.1 Detailed Description

Simple class to remote control the super tux game. Class can be inherited to steer the penguin. TuxControlSingleton class has to be adapted, that it creates an instance of the right descendant of TuxControl. Since TuxControl itself offers all the necessary methods, TuxControlSingleton could also be used with the raw TuxControl implementation In that case the same instance of TuxControl (via TuxControlSingleton) would had to be aggregated from a separate thread to control the robot. With a mutex TuxControl is thread safe. Author: Lars Widmer, www.lawi.ch

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ClassificationTux::ClassificationTux ()

Constructor: Initializes the object and all its aggregations. The class is a decendant from TuxControl. Uses the LearningAlgorithm implementation NearestClusterCenter to control super tux by gesture recognition with sensors. For the classification running in parallel a thread is started.

3.1.2.2 ClassificationTux::~ClassificationTux () [virtual]

Stops the classification thread and cleans up.

3.1.3 Member Function Documentation

3.1.3.1 void ClassificationTux::threadMethod () [virtual]

Actual thread method calling a classification method like continous or segmented and evaluate to generate the key events.

Reimplemented from Thread.

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

- ClassificationTux.h
- ClassificationTux.cpp

3.2 Cloud Struct Reference

`#include <NearestClusterCenter.h>`

Public Attributes

- std::vector< Features > **data**
- OutputType **value**

3.2.1 Detailed Description

Class for learning and classification based on feature distances. Algorithm is the nearest cluster center.

Author:

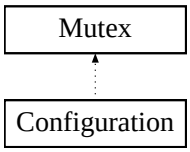
Lars Widmer www.lawi.ch

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

- NearestClusterCenter.h

3.3 Configuration Class Reference

`#include <Configuration.h>`Inheritance diagram for Configuration::



Public Member Functions

- virtual ~Configuration ()
- std::string getFilename ()
- void clear ()
- void **clear** (std::string)
- void **remove** (std::string)
- std::vector< std::string > **getStrings** (std::string)
- std::string **getString** (std::string)
- std::vector< int > **getInts** (std::string)
- int **getInt** (std::string)
- void **set** (std::string, std::vector< std::string >)
- void **set** (std::string, std::string)
- void **set** (std::string, std::vector< int >)
- void **set** (std::string, int)
- void **set** (std::string)
- void **add** (std::string, std::vector< std::string >)
- void **add** (std::string, std::string)
- void **add** (std::string, std::vector< int >)
- void **add** (std::string, int)
- void **add** (std::string)
- void **load** (std::string)
- void **save** (std::string)
- ISVec getISVec ()
- void print ()

Static Public Member Functions

- static Configuration * **getInstance** (std::string="newconfig.xml")
- static void **createFile** (std::string)

3.3.1 Detailed Description

Configuration.h Class offers access to configuration data stored in XML files. The system is flexible and not bound to a single application. Possible data formats are:

- int
- string

- vector<int>
- vector<string> The values are identified by a string name. A generic configure application allows to edit the configuration as it's stored in file. But Due to the human readable xml file it's often simpler and faster to edit the file using a standard text editor.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Configuration::~Configuration () `[virtual]`

Empty destructor.

3.3.3 Member Function Documentation

3.3.3.1 void Configuration::clear ()

Clears the internal configuration cache.

3.3.3.2 std::string Configuration::getFilename ()

Returns the name of the loaded file.

3.3.3.3 ISVec Configuration::getISVec ()

Returns the internal cache data structure. For most usages it shouldn't be necessary to use this function.

3.3.3.4 void Configuration::print ()

Prints the configuration data in a simple way.

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

- Configuration.h
- Configuration.cpp

3.4 ControlEvent Struct Reference

Public Attributes

- bool **valid**
- int **key**
- int **event**

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

- TuxControl.h

3.5 DataStrings Class Reference

Static Public Member Functions

- static int getNibbles (int, string, int)
- static int getBytes (int, string, int)
- static unsigned getByte (string, int)
- static int getWord (string, int)
- static unsigned int decodeByte (string code)
- static string encodeByte (unsigned int num)
- static int decodeWord (string code)
- static string encodeWord (int num)
- static int decode (unsigned bit, string code)
- static string encode (unsigned bit, int num)
- static string encodeFloat (float)
- static float decodeFloat (string)
- static vector< NodeAndData > getData (string)
- static string getDataString (NodeAndData)
- static string getDataString (NodeAndData, int)
- static string getDataString (vector< NodeAndData >, int)
- static string encodeLimbPos (LimbNameCoordinates)
- static LimbNameCoordinates decodeLimbPos (string)
- static string roggenEncode (vector< int >)
- static vector< int > roggenDecode (string)

3.5.1 Member Function Documentation

3.5.1.1 int DataStrings::decode (unsigned *bit*, string *code*) [static]

Decodes a given string of a given number of bytes. Every character gets decoded into 4 bits. Therefore the given number of bits has to be a multiple of 4.

3.5.1.2 unsigned int DataStrings::decodeByte (string *code*) [static]

Helper method to decode a byte for sending as a text. In the input string only the characters from A to P can be used. Therefore one byte is stored in two characters.

3.5.1.3 float DataStrings::decodeFloat (string *code*) [static]

Helper method to decode a small float after receiving it as a text. We receive 16bit. Legal are values between -30.0 and 30.0. We deal with 3 digits after the comma.

3.5.1.4 LimbNameCoordinates DataStrings::decodeLimbPos (string *data*) [static]

Extracts the name and position of a limb out of the given string.

3.5.1.5 int DataStrings::decodeWord (string *code*) [static]

Helper method to decode a word after receiving it as a text. We assume 16bit values. This is what the standard says about the minimum size of int. Therefore the function exits if there accur larger values. This function returns values from -32768 to 32767.

3.5.1.6 string DataStrings::encode (unsigned *bit*, int *num*) [static]

Encodes a given value of a given number of bytes. Every 4 bits are encoded in one character. Therefore the given number of bits has to be a multiple of 4.

3.5.1.7 string DataStrings::encodeByte (unsigned int *num*) [static]

Helper method to encode a byte for sending as a text. In the output string only the characters from A to P are used. Therefore one byte takes two characters to store. In return we have loads of escape characters left and we don't have to be afraid of eof occuring in the data stream. Values bigger then 255 aren't accepted. In other words only the lowest 8 Bits are wanted. Unsigned numbers are expected!

3.5.1.8 string DataStrings::encodeFloat (float *num*) [static]

Helper method to encode a small float number for sending as a text. We send 16bit. Legal are values between -30.0 and 30.0. The function exits if there accur larger values. We transmit 3 digits after the comma.

3.5.1.9 string DataStrings::encodeLimbPos (LimbNameCoordinates *data*) [static]

Encodes the name and position of a limb into a string.

3.5.1.10 string DataStrings::encodeWord (int *num*) [static]

Helper method to encode a word for sending as a text. We assume 16bit values. This is what the standard says about the minimum size of int. Therefore the function exits if there accur larger values. This function handles values from -32768 to 32767.

3.5.1.11 unsigned DataStrings::getByte (string *str*, int *start*) [static]

Extracts an unsigned number ouf of a string. The length is fixed to one byte. Therefore the number is between 0 and 255. In the string this is a value between AA and PP.

3.5.1.12 int DataStrings::getBytes (int *numOfBytes*, string *str*, int *start*) [static]

Extracts a number of bytes out of a string. Every byte there is encoded into two characters from A to P. This method returns positive and negative numbers.

3.5.1.13 vector< NodeAndData > DataStrings::getData (string *data*) [static]

Extracts the sensor data out of the given string.

3.5.1.14 string DataStrings::getDataString (vector< NodeAndData > *nads*, int *id*) [static]

Helper method

3.5.1.15 string DataStrings::getDataString (NodeAndData *nad*, int *id*) [static]

Helper method

3.5.1.16 string DataStrings::getDataString (NodeAndData *nad*) [static]

Helper method

3.5.1.17 int DataStrings::getNibbles (int *numOfNibbles*, string *str*, int *start*) [static]

Extracts a number of nibbles (half bytes; 4 bits) out of a string. Every byte there is encoded into two characters from A to P. This method returns positive and negative numbers.

3.5.1.18 int DataStrings::getWord (string *str*, int *start*) [static]

Extracts a 2 byte value out of a string. Every byte there is encoded into two characters. Therefore the given string must be four characters long. This method returns positive and negative numbers.

3.5.1.19 vector< int > DataStrings::roggenDecode (string *str*) [static]

Decodes a string back to vector of int (used as stream frame for the roggen sensors).

3.5.1.20 string DataStrings::roggenEncode (vector< int > *data*) [static]

Encodes a vector of int (used as stream frame for the roggen sensors) into a string.

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

- DataStrings.h
- DataStrings.cpp

3.6 Defines Class Reference

`#include <Defines.h>`

3.6.1 Detailed Description

No code in this class, just the define statements. Author: Lars;

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

- Defines.h

3.7 EulerRotation Struct Reference

Public Attributes

- float **xRotation**
- float **yRotation**
- float **zRotation**

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

- Defines.h

3.8 FeatureSample Struct Reference

Public Attributes

- Features **in**
- OutputType **out**

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

- LearningAlgorithm.h

3.9 FloatData Struct Reference

Public Attributes

- FSVec **dats**
- std::string **name**

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

- XmlFileHandling.h

3.10 FloatStruct Struct Reference

Public Attributes

- FloatVec **vals**
- StringVec **defs**
- std::string **name**
- float **num**

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

- XmlFileHandling.h

3.11 FrameParser2 Class Reference

Public Member Functions

- **FrameParser2** (std::string _format)
- void **Status** ()
- std::vector< std::vector< int > > **Parser** (const char *data, int n)
- bool **IsValid** ()
- int **GetFrameSize** ()
- int **GetNumChannels** ()

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

- FrameParser2.h
- FrameParser2.cpp

3.12 GestureManagement Class Reference

Public Member Functions

- GestureManagement ()
- virtual ~GestureManagement ()
- bool checkAllFiles ()
- void updateBuffers ()
- void fillBuffer ()
- bool training ()
- void wait ()
- void reenter ()
- int numberOfSensors ()
- void setBufferToEnergyFrameSize (int)
- void setBufferToEnergyFrameSize ()
- void setBufferToLongestGestureSize (int)
- void setBufferToLongestGestureSize ()
- bool bufferFull (int)
- bool bufferFull ()
- bool gotEnergy (int)
- bool gotEnergy ()
- bool lostEnergy (int)
- bool lostEnergy ()
- void setBufferUnlimited (int)
- void setBufferUnlimited ()
- bool isSensorTrained (int)
- std::pair< Output, FeatureType > classify (int)
- std::pair< Output, FeatureType > classify (int, int)
- bool isGestureLengthOk (int, int)
- int sensorOfGesture (int)
- bool continousMode ()
- int numberOfGestures ()
- void manageGestures ()

3.12.1 Constructor & Destructor Documentation

3.12.1.1 GestureManagement::GestureManagement ()

Initialize and prepare the aggregated instances.

3.12.1.2 GestureManagement::~GestureManagement () `[virtual]`

Clean up and delete the the aggregated instances.

3.12.2 Member Function Documentation

3.12.2.1 bool GestureManagement::bufferFull ()

Method used by the ClassificationTux class. Returns true if all buffers are full and serve as a queue.

3.12.2.2 bool GestureManagement::bufferFull (int *sensor*)

Method used by the ClassificationTux class. Returns true if the buffer of the given sensor is full and servers as a queue.

3.12.2.3 bool GestureManagement::checkAllFiles ()

Returns true if all necessary files for training are available.

3.12.2.4 pair< Output, FeatureType > GestureManagement::classify (int *sensor*, int *gesture*)

Method used by the ClassificationTux class. Calls the classification function of the learning algorithm for the size of the given gesture. This method is used for continous classification.

3.12.2.5 pair< Output, FeatureType > GestureManagement::classify (int *sensor*)

Method used by the ClassificationTux class. Calls the classification function of the learning algorithm.

3.12.2.6 bool GestureManagement::continousMode ()

Method used by the ClassificationTux class. Returns true if the mode is set to continous classification. This can be set in the configuration file.

3.12.2.7 void GestureManagement::fillBuffer ()

Read and store data until the buffer is full.

3.12.2.8 bool GestureManagement::gotEnergy ()

Method used by the ClassificationTux class. Returns true when all sensors (0, 1, ...) are active.

3.12.2.9 bool GestureManagement::gotEnergy (int *sensor*)

Method used by the ClassificationTux class. Returns true when the given sensor (0, 1, ...) is active.

3.12.2.10 bool GestureManagement::isGestureLengthOk (int *sensor*, int *gesture*)

Method used by the ClassificationTux class. Checks if the gesture length we got by energy based segmentation is in the same range as the recorded movements for this gestures. This method is used for segmented classification.

3.12.2.11 bool GestureManagement::isSensorTrained (int *sensor*)

Method used by the ClassificationTux class. Returns true if the given sensor is used for recognition and has been trained.

3.12.2.12 bool GestureManagement::lostEnergy ()

Method used by the ClassificationTux class. Returns true when all sensors (0, 1, ...) are inactive.

3.12.2.13 bool GestureManagement::lostEnergy (int *sensor*)

Method used by the ClassificationTux class. Returns true when the given sensor (0, 1, ...) is inactive.

3.12.2.14 void GestureManagement::manageGestures ()

Highest level function. Calls mainLoop.

3.12.2.15 int GestureManagement::numberOfGestures ()

Method used by the ClassificationTux class. Returns the number of gestures. Gestures are numbered from zero to number of gestures - 1.

3.12.2.16 int GestureManagement::numberOfSensors ()

Method used by the ClassificationTux class. Returns the number of sensors used as defined in the configuration file.

3.12.2.17 void GestureManagement::reenter ()

Wait until all sensors are inactive.

3.12.2.18 int GestureManagement::sensorOfGesture (int *gesture*)

Method used by the ClassificationTux class. Returns the number of the sensor for a given gesture number. Both are numbered internally from zero onwards. Basically this method accesses the activity information. For a given gesture it tells which sensor is the active one.

3.12.2.19 void GestureManagement::setBufferToEnergyFrameSize ()

Method used by the ClassificationTux class. Sets all buffers to the size for energy based segmentation. This size can be configured in the configuration file.

3.12.2.20 void GestureManagement::setBufferToEnergyFrameSize (int *sensor*)

Method used by the ClassificationTux class. Sets the buffer of the given sensor to the size for energy based segmentation. This size can be configured in the configuration file.

3.12.2.21 void GestureManagement::setBufferToLongestGestureSize ()

Method used by the ClassificationTux class. Sets all buffers to the size long enough that even the longest recorded gesture will fit inside. Before this method readData or training have to be called.

3.12.2.22 void GestureManagement::setBufferToLongestGestureSize (int *sensor*)

Method used by the ClassificationTux class. Sets the buffer of the given sensor to the size long enough that even the longest recorded gesture fits inside. Before this method readData or training have to be called.

3.12.2.23 void GestureManagement::setBufferUnlimited ()

Method used by the ClassificationTux class. Sets all buffers to unlimited size. So they will never remove frames from the queue.

3.12.2.24 void GestureManagement::setBufferUnlimited (int *sensor*)

Method used by the ClassificationTux class. Sets the buffer of the given sensor (0,1,...) to unlimited size. So it will never remove frames from the queue.

3.12.2.25 bool GestureManagement::training ()

High level method calls all necessary methods for training the algorithm. First the gesture data is loaded from disk. Then inactive sensors are removed and the algorithm gets trained.

3.12.2.26 void GestureManagement::updateBuffers ()

Read data from the sensors and put it to the buffers.

3.12.2.27 void GestureManagement::wait ()

Wait for the user hitting a key while still updating the buffer with current sensor data.

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

- GestureManagement.h
- GestureManagement.cpp

3.13 Helper Class Reference

Public Member Functions

- Helper ()
- unsigned long int uSecondsSinceStart ()
- void restartuSeconds ()
- void setRowWidth (const int &)
- template<class T >
  string **inRow** (const T &)
- template<class T >
  string **inRow** (const vector< T > &)
- template<class T >
  string **inRow** (const T &, const int &)
- template<class T >
  void **echo** (const T &)
- template<class T >
  void **echo** (const T &, const T &)
- template<class T >
  void **echo** (const T &, const T &, const T &)
- template<class T >
  void **echo** (const vector< T > &)
- template<class T >
  void **echoBr** (const T &)
- template<class T >
  void **echoBr** (const T &, const T &)
- template<class T >
  void **echoBr** (const T &, const T &, const T &)
- template<class T >
  void **echoBr** (const vector< T > &)
- void setRowSize (const int &)

Static Public Member Functions

- static int getch ()
- static int kbhit ()
- static Helper * **getInstance** ()
- static string str2UpperCase (const string &)
- static void specialSleep (const unsigned &)
- static pair< unsigned long int, unsigned long int > secAnduSec ()
- static unsigned long int uSeconds ()
- static unsigned long int seconds ()
- static char num2Char (int)
- template<class in_value , class out_value >
  static void **convert** (const in_value &ival, out_value &oval)
- template<class T >
  static string **toString** (const T &)
- template<class T >
  static T **abs** (const T &)
- template<class T >
  static T **getMax** (const T &, const T &)

- template<class T >
  static bool **has** (const vector< T > &, const T &)
- template<class T >
  static int **locate** (const vector< T > &, const T &)
- template<class T >
  static vector< T > **sort** (const vector< T > &)
- static LimbCoordinates vectorDifference (const LimbCoordinates &, const LimbCoordinates &)
- static LimbCoordinates vectorSum (const LimbCoordinates &, const LimbCoordinates &)
- static EulerRotation vectorAngles (const LimbCoordinates &)
- static float vectorLength (const LimbCoordinates &)
- static float scalarProduct (const LimbCoordinates &, const LimbCoordinates &)
- static LimbCoordinates normalize (const LimbCoordinates &)
- static LimbCoordinates multiply (const LimbCoordinates &, float)
- static LimbCoordinates vectorProduct (const LimbCoordinates &, const LimbCoordinates &)
- static float angleBetweenVectors (const LimbCoordinates &, const LimbCoordinates &)
- static vector< float > vectorAngles2 (const LimbCoordinates &)
- static LimbCoordinates rotate (float, float, float, float, float, float)
- static LimbCoordinates rotate (LimbCoordinates, float, float, float)
- static LimbCoordinates rotate (LimbCoordinates, EulerRotation)
- static vector< float > unitCirclePoint (vector< float >)
- static vector< float > unitCirclePoint (EulerRotation)
- static vector< float > unitCirclePoint (vector< float >, float)
- static float unitCircleDistance (vector< float >, vector< float >)
- static void print2DecimalPlaces (const vector< float > &)
- static bool fileExists (const char *)
- template<class T >
  static void **checkSize** (const vector< T > &, const vector< T > &)
- template<class T >
  static vector< T > **vectorDifference** (const vector< T > &, const vector< T > &)
- template<class T >
  static vector< T > **vectorSum** (const vector< T > &, const vector< T > &)
- template<class T >
  static float **vectorLength** (const vector< T > &)
- template<class T >
  static vector< double > **normalize** (const vector< T > &)
- template<class T >
  static float **scalarProduct** (const vector< T > &, const vector< T > &)
- template<class T >
  static float **angleBetweenVectors** (const vector< T > &, const vector< T > &)
- template<class T >
  static void **print** (const vector< T > &)
- template<class T >
  static string **toString** (const vector< T > &)
- template<class T >
  static bool **equal** (const vector< T > &, const vector< T > &)

3.13.1 Constructor & Destructor Documentation

3.13.1.1 Helper::Helper ()

Constructor The class can be used with or without constructor. Both is possible. The echo fetures are only available on a valid instance. This can be done via getInstance or the constructor.

3.13.2 Member Function Documentation

3.13.2.1 float Helper::angleBetweenVectors (const LimbCoordinates & *vec1*, const LimbCoordinates & *vec2*) [static]

Returns the angle between the two vectors. We assume all components of vec1 always to be positive. Works only for outputs in the range of 0..PI. There is no signum set.

3.13.2.2 bool Helper::fileExists (const char * *filename*) [static]

Returns true if the file at the given path exists.

3.13.2.3 int Helper::getch (void) [static]

getch() -- a blocking single character input from stdin

Returns a character, or -1 if an input error occurs.

Conditionals allow compiling with or without echoing of the input characters, and with or without flushing pre-existing existing buffered input before blocking.

3.13.2.4 int Helper::kbhit (void) [static]

kbhit() -- a keyboard lookahead monitor

returns the number of characters available to read.

3.13.2.5 LimbCoordinates Helper::multiply (const LimbCoordinates & *vec*, float *factor*) [static]

Returns the given vector streched by the given factor. result = vec1 * factor

3.13.2.6 LimbCoordinates Helper::normalize (const LimbCoordinates & *vec*) [static]

Normalizes the given vector to the length 1.

3.13.2.7 char Helper::num2Char (int *num*) [static]

Converts a integer number to an ascii character. There is no range checking. The intention is only to use numbers between 1 and 24. 1 gets to A.

3.13.2.8 void Helper::print2DecimalPlaces (const vector< float > & *vec*) [static]

Prints the given vector to standard output.

3.13.2.9 void Helper::restartuSeconds ()

Resets the starting point for uSecondsSinceStart(). Since the class is a singleton the reset applies for any usage!

3.13.2.10 LimbCoordinates Helper::rotate (LimbCoordinates *c*, EulerRotation *r*) [static]

Rotates the point at the given coordinate. This method rotates first around the x, then y and finally around the z axis. The new coordinates are returned.

3.13.2.11 LimbCoordinates Helper::rotate (LimbCoordinates *c*, float *r1*, float *r2*, float *r3*) [static]

Rotates the point at the given coordinate. This method rotates first around the x, then y and finally around the z axis. The new coordinates are returned.

3.13.2.12 LimbCoordinates Helper::rotate (float *x*, float *y*, float *z*, float *r1*, float *r2*, float *r3*) [static]

Rotates the point at the given coordinate. This method rotates first around the x, then y and finally around the z axis. The new coordinates are returned.

3.13.2.13 float Helper::scalarProduct (const LimbCoordinates & *vec1*, const LimbCoordinates & *vec2*) [static]

Returns the scalar product of the two given vectors.

3.13.2.14 pair< unsigned long int, unsigned long int > Helper::secAnduSec () [static]

Returns a pair of two numbers. The first one tells the seconds since January 1, 1970. The second number tells the microseconds since the last full second.

3.13.2.15 unsigned long int Helper::seconds () [static]

Returns the seconds since January 1, 1970.

3.13.2.16 void Helper::setRowSize (const int & *newSize*)

Sets the row width for inRow, echoBr ... Synonym to setRowWidth.

3.13.2.17 void Helper::setRowWidth (const int & *newEchoMax*)

Sets the row width for inRow, echoBr ...

3.13.2.18 void Helper::specialSleep (const unsigned & *microseconds*) [static]

Simple platform independent sleep function. Author: Lars

3.13.2.19 string Helper::str2UpperCase (const string & *str*) [static]

Static helper method. Returns the given string in upper case.

3.13.2.20 float Helper::unitCircleDistance (vector< float > *py1*, vector< float > *py2*) [static]

Returns the distance of two pairs of pitch and yaw by interpreting the given values as points on the unit circle.

3.13.2.21 vector< float > Helper::unitCirclePoint (vector< float > *py*, float *radius*) [static]

Returns a point (x,y,z) on the unit circle for the given values of pitch an yaw.

3.13.2.22 vector< float > Helper::unitCirclePoint (EulerRotation *er*) [static]

Returns a point (x,y,z) on the unit circle for the given values of pitch an yaw.

3.13.2.23 vector< float > Helper::unitCirclePoint (vector< float > *py*) [static]

Returns a point (x,y,z) on the unit circle for the given values of pitch an yaw.

3.13.2.24 unsigned long int Helper::uSeconds () [static]

Returns the microseconds since the last full second of seconds().

3.13.2.25 unsigned long int Helper::uSecondsSinceStart ()

Returns the microseconds since the program has been started. This is comfortable but after running the program for more than an hour an overflow occurs. So be aware of this!

3.13.2.26 EulerRotation Helper::vectorAngles (const LimbCoordinates & *vec*) [static]

Returns the euler angles of the given vector. Depends on the method angleBetweenVectors. The y rotation (roll) of the return value is always zero because there is no roll for a vector.

3.13.2.27 vector< float > Helper::vectorAngles2 (const LimbCoordinates & *vec*) [static]

Returns the pitch and yaw (azimuth) of the given vector. Depends on the method vectorAngles. The y rotation (roll) of the return value is always zero because there is no roll for a vector.

3.13.2.28 LimbCoordinates Helper::vectorDifference (const LimbCoordinates & *vec1*, const LimbCoordinates & *vec2*) [static]

Returns the difference vector of the two given vectors. result = vec1 - vec2

3.13.2.29 float Helper::vectorLength (const LimbCoordinates & *vec*) [static]

Returns the length of the given vector.

3.13.2.30 LimbCoordinates Helper::vectorProduct (const LimbCoordinates & *vec1*, const LimbCoordinates & *vec2*) [static]

Find vector product of the two given vectors.

3.13.2.31 LimbCoordinates Helper::vectorSum (const LimbCoordinates & *vec1*, const LimbCoordinates & *vec2*) [static]

Returns the sum vector of the two given vectors. result = vec1 + vec2

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

- Helper.h
- Helper.cpp

3.14 IntData Struct Reference

Public Attributes

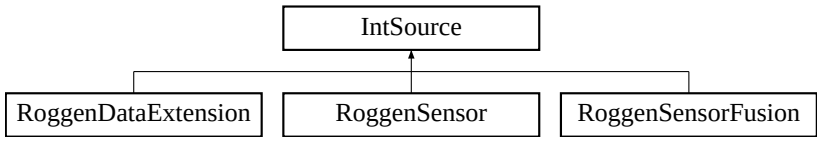
- ISVec **dats**
- std::string **name**

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

- XmlFileHandling.h

3.15 IntSource Class Reference

`#include <IntSource.h>`Inheritance diagram for IntSource::



Public Member Functions

- virtual std::vector< std::vector< int > > **getData** ()=0
- virtual ~IntSource ()

3.15.1 Detailed Description

Interface class: Every descendant has to offer a getData function. Example descendants are the sensor classes. Author: Lars Widmer, www.lawi.ch

3.15.2 Constructor & Destructor Documentation

3.15.2.1 IntSource::~IntSource () `[virtual]`

Empty destructor (virtual). Class is just an interface.

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

- IntSource.h
- IntSource.cpp

3.16 IntStruct Struct Reference

Public Attributes

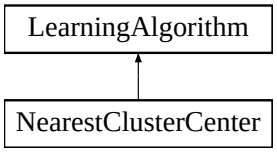
- IntVec **vals**
- StringVec **defs**
- std::string **name**
- int **num**

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

- XmlFileHandling.h

3.17 LearningAlgorithm Class Reference

Inheritance diagram for LearningAlgorithm::



Public Member Functions

- virtual ~LearningAlgorithm ()
- virtual void **reset** ()=0
- virtual void **train** (FeatureSet)=0
- virtual Output **classify** (Features)=0
- virtual std::pair< Output, FeatureType > **classification** (Features)=0
- virtual void **setUnknownThreshold** (float)=0
- virtual float **getUnknownThreshold** ()=0
- virtual void **print** ()=0

3.17.1 Constructor & Destructor Documentation

3.17.1.1 LearningAlgorithm::~LearningAlgorithm () `[virtual]`

Pure interface class. Empty (virtual) destructor.

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

- LearningAlgorithm.h
- LearningAlgorithm.cpp

3.18 LimbCoordinates Struct Reference

Public Attributes

- float **x**
- float **y**
- float **z**

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

- Defines.h

3.19 LimbNameCoordinates Struct Reference

Public Attributes

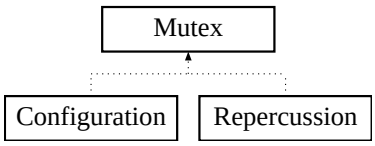
- std::string **name**
- float **x**
- float **y**
- float **z**

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

- Defines.h

3.20 Mutex Class Reference

`#include <Mutex.h>`Inheritance diagram for Mutex::



Public Member Functions

- Mutex ()
- virtual ~Mutex ()
- void acquireMutex ()
- void releaseMutex ()

3.20.1 Detailed Description

Simple class for inheritance or aggregation. It provides a single mutex (mutal exclusion). Using it as a super class is easier. Whereas aggregation has to be used in the case when more than one mutex is needed. Author: Lars Widmer, www.lawi.ch

3.20.2 Constructor & Destructor Documentation

3.20.2.1 Mutex::Mutex ()

Constructor: Initializes the mutual exclusion.

3.20.2.2 Mutex::~Mutex () `[virtual]`

Destructor: Destroys the mutual exclusion.

3.20.3 Member Function Documentation

3.20.3.1 void Mutex::acquireMutex ()

Acquire the mutual exclusion.

3.20.3.2 void Mutex::releaseMutex ()

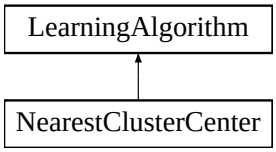
Release the mutual exclusion.

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

- Mutex.h
- Mutex.cpp

3.21 NearestClusterCenter Class Reference

Inheritance diagram for NearestClusterCenter::



Public Member Functions

- NearestClusterCenter ()
- NearestClusterCenter (FeatureSet)
- virtual ~NearestClusterCenter ()
- virtual void reset ()
- virtual void train (FeatureSet)
- virtual Output classify (Features)
- virtual std::pair< Output, FeatureType > classification (Features)
- virtual void setUnknownThreshold (float)
- virtual float getUnknownThreshold ()
- virtual void print ()

3.21.1 Constructor & Destructor Documentation

3.21.1.1 NearestClusterCenter::NearestClusterCenter ()

Constructor: Clears all the internal fields and leaves the algorithm untrained.

3.21.1.2 NearestClusterCenter::NearestClusterCenter (FeatureSet *s*)

Constructor: Clears all the internal fields and trains the algorithm with the given set of features.

3.21.1.3 NearestClusterCenter::~NearestClusterCenter () `[virtual]`

Empty Destructor

3.21.2 Member Function Documentation

3.21.2.1 pair< Output, FeatureType > NearestClusterCenter::classification (Features *in*) `[virtual]`

Classification function. Matches the given point in feature space to the nearest cluster center. The classification is marked valid depending if the distance between the point and the center is less then the defined threshold. The first value of the output pair is the found classification in an Output struct. The Output struct contains a flag if the classification is valid and the value of the classification. The second value of the output pair tells the distance between the given point and the matched cluster center.

Implements LearningAlgorithm.

3.21.2.2 Output NearestClusterCenter::classify (Features *in*) [virtual]

Classification function. Matches the given point in feature space to the nearest cluster center. The classification is marked valid depending if the distance between the point and the center is less then the defined threshold. The return value is the found classification as an Output struct. The Output struct contains a flag if the classification is valid and the value of the classification.

Implements LearningAlgorithm.

3.21.2.3 float NearestClusterCenter::getUnknownThreshold () [virtual]

Getter function for the classification threshold. If a point in feature space is classified with a distance from the nearest cluster bigger then the threshold it's marked as invalid.

Implements LearningAlgorithm.

3.21.2.4 void NearestClusterCenter::print () [virtual]

Print function for the set of cluster centers.

Implements LearningAlgorithm.

3.21.2.5 void NearestClusterCenter::reset () [virtual]

Clears the internal fields e.g. resets the training state to untrained.

Implements LearningAlgorithm.

3.21.2.6 void NearestClusterCenter::setUnknownThreshold (float *newUT*) [virtual]

Setter function for the classification threshold. If a point in feature space is classified with a distance from the nearest cluster bigger then the threshold it's marked as invalid.

Implements LearningAlgorithm.

3.21.2.7 void NearestClusterCenter::train (FeatureSet *set*) [virtual]

Trains the algorithm with the given training set.

Implements LearningAlgorithm.

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

- NearestClusterCenter.h
- NearestClusterCenter.cpp

3.22 Neighbour Struct Reference

Public Attributes

- CoordSet **positions**
- std::string **name**

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

- XmlFileHandling.h

3.23 NodeAndData Struct Reference

Public Attributes

- NodeData **data**
- int **node**

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

- Defines.h

3.24 NodeData Struct Reference

Public Attributes

- float **accX**
- float **accY**
- float **accZ**
- float **magX**
- float **magY**
- float **magZ**
- float **volts**
- bool **valid**

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

- Defines.h

3.25 Output Struct Reference

Public Attributes

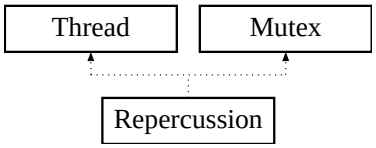
- OutputType **value**
- bool **valid**

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

- LearningAlgorithm.h

3.26 Repercussion Class Reference

`#include <Repercussion.h>`Inheritance diagram for Repercussion::



Public Member Functions

- Repercussion ()
- Repercussion (int)
- virtual ~Repercussion ()
- void threadMethod ()
- void set (int, int=1)
- int get (int)
- void resetAll ()
- void reset (int)
- void inc (int, int=1)
- void dec (int, int=1)
- void incAll (int=1)
- void setAll (int=1)
- void decAll (int=1)
- std::vector< int > getAll ()
- bool isZero (int)
- void activate ()
- void deactivate ()
- void setDelay (int)
- void setDecrement (int)
- void setMaximum (int)

3.26.1 Detailed Description

Class for managing values which decrease against time. It can be used to make single events lasting over a defined amount of time. Author: Lars Widmer, www.lawi.ch

3.26.2 Constructor & Destructor Documentation

3.26.2.1 Repercussion::Repercussion ()

Constructor creates and initializes an empty instance.

3.26.2.2 Repercussion::Repercussion (int *number*)

Constructor creates and initializes an instance for the use with the given number of values.

3.26.2.3 Repercussion::~Repercussion () [virtual]

Destructor: Stops the internal thread.

3.26.3 Member Function Documentation

3.26.3.1 void Repercussion::activate ()

Activates the repercussion instance. Initially the instance of this class is inactive. It doesn't start decrementing before it's activated. First define all the settings and then activate.

3.26.3.2 void Repercussion::deactivate ()

Deactivates the repercussion instance. It doesn't decrement while inactive.

3.26.3.3 void Repercussion::dec (int *val*, int *amount* = 1)

Subtract the amount given in the second parameter from the value with the index given in the first parameter. With this method the values can't get negative.

3.26.3.4 void Repercussion::decAll (int *amount* = 1)

Subtract the given amount from all values. With this method the values can't get negative.

3.26.3.5 int Repercussion::get (int *val*)

Return the value with the given index.

3.26.3.6 std::vector< int > Repercussion::getAll ()

Returns all the values as a vector.

3.26.3.7 void Repercussion::inc (int *val*, int *amount* = 1)

Add the amount given in the second parameter to the value with the index given in the first parameter. With this method the values can't go over the defined maximum.

3.26.3.8 void Repercussion::incAll (int *amount* = 1)

Add the given amount to all values. With this method the values can't go over the defined maximum.

3.26.3.9 bool Repercussion::isZero (int *val*)

Returns true if the value with the given index is zero (or below).

3.26.3.10 void Repercussion::reset (int *val*)

Reset the value with the given index number to zero.

3.26.3.11 void Repercussion::resetAll ()

Reset all values to zero.

3.26.3.12 void Repercussion::set (int *val*, int *level* = 1)

Set the value with the given index to the given level.

3.26.3.13 void Repercussion::setAll (int *level* = 1)

Set all values to the given level.

3.26.3.14 void Repercussion::setDecrement (int *subtr*)

Setter function for the amount to subtract from the values in each step. Default is 1.

3.26.3.15 void Repercussion::setDelay (int *uSec*)

Setter function for the delay in microseconds between the decrementation steps. Use values above 1000. Default is 1000.

3.26.3.16 void Repercussion::setMaximum (int *max*)

Setter function for the maximum value. Default is 500.

3.26.3.17 void Repercussion::threadMethod () [virtual]

Thread method aging the stored values according to the defined settings.

Reimplemented from Thread.

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

- Repercussion.h
- Repercussion.cpp

3.27 RoggenBuffer Class Reference

Public Member Functions

- RoggenBuffer ()
- RoggenBuffer (std::vector< std::vector< int > >)
- RoggenBuffer (int)
- virtual ~RoggenBuffer ()
- std::vector< std::vector< int > > * getPointer ()
- std::vector< std::vector< int > > getRange (int, int)
- std::vector< std::vector< int > > get ()
- std::vector< std::vector< int > > getLast ()
- bool isFresh ()
- void clear ()
- void setLength (int)
- int size ()
- int maxSize ()
- bool isFull ()
- std::vector< int > at (int)
- void **put** (std::vector< int >)
- void **put** (std::vector< std::vector< int > >)
- std::vector< int > getSingleMeasurement (int)
- std::vector< std::vector< int > > getFrames (int, int)
- void checkSize (int)
- void check (int, int)
- std::vector< std::vector< int > > filter (int, int)
- void filterPut (int, int, std::vector< int >)
- void filterPut (int, int, std::vector< std::vector< int > >)
- void setUnlimited ()
- void resetUnlimited ()
- void setUnlimited (bool flag)

Static Public Member Functions

- static std::vector< int > **getSingleMeasurement** (int, std::vector< std::vector< int > >)
- static std::vector< std::vector< int > > **getFrames** (int, int, std::vector< std::vector< int > >)
- static std::vector< int > **getSum** (std::vector< int >, std::vector< int >)
- static std::vector< int > **getSum** (std::vector< int >, std::vector< int >, std::vector< int >)
- static std::vector< int > **getVectorialSum** (std::vector< int >, std::vector< int >)
- static std::vector< int > **getVectorialSum** (std::vector< int >, std::vector< int >, std::vector< int >)
- static void checkSize (int, int)
- static void check (int, int, int)
- static std::vector< std::vector< int > > **filter** (int, int, std::vector< std::vector< int > > *)
- static std::vector< std::vector< int > > filter (int, int, std::vector< std::vector< int > >)

3.27.1 Constructor & Destructor Documentation

3.27.1.1 RoggenBuffer::RoggenBuffer ()

Constructor: Initializes the class fields and sets the buffer to default size.

3.27.1.2 RoggenBuffer::RoggenBuffer (std::vector< std::vector< int > > *data*)

Constructor: Initializes the class fields and sets the buffer to default size. The given data is stored in the buffer.

3.27.1.3 RoggenBuffer::RoggenBuffer (int *bufferSize*)

Constructor: Initializes the class fields and sets the buffer to the given size.

3.27.1.4 RoggenBuffer::~RoggenBuffer () `[virtual]`

Destructor: Currently nothing to do.

3.27.2 Member Function Documentation

3.27.2.1 vector< int > RoggenBuffer::at (int *index*)

Returns the frame at the given buffer position.

3.27.2.2 void RoggenBuffer::check (int *start*, int *end*)

Returns true if the given two numbers both are above zero and below or equal to the buffer size.

3.27.2.3 void RoggenBuffer::check (int *start*, int *end*, int *max*) `[static]`

Returns true if the given first two numbers both are above zero and below or equal to the third parameter.

3.27.2.4 void RoggenBuffer::checkSize (int *num*)

Returns true if the given number is above zero and below or equal to the buffer size.

3.27.2.5 void RoggenBuffer::checkSize (int *num*, int *max*) `[static]`

Returns true if the given first number is above zero and below or equal to the second parameter.

3.27.2.6 void RoggenBuffer::clear ()

Clears the internal buffer.

3.27.2.7 vector< vector< int > > RoggenBuffer::filter (int *meas*, int *val*)

Filters the internal buffer. The measurement with the given index must have the given value. Frames which satisfy this condition are put to the result vector.

3.27.2.8 vector< vector< int > > RoggenBuffer::filter (int *meas*, int *val*, std::vector< std::vector< int > > *buf*) [static]

Filters the data from the given buffer. The measurement with the given index must have the given value. Frames which satisfy this condition are put to the result vector.

3.27.2.9 void RoggenBuffer::filterPut (int *meas*, int *val*, std::vector< std::vector< int > > *lines*)

Filters and puts the given buffer. The measurement with the given index must have the given value. The given frames which satisfy this condition are put to the internal buffer.

3.27.2.10 void RoggenBuffer::filterPut (int *meas*, int *val*, std::vector< int > *line*)

Filters and puts the given frame. The measurement with the given index must have the given value. If the given frame satisfies this condition it's put to the internal buffer.

3.27.2.11 vector< vector< int > > RoggenBuffer::get ()

Getter function for the whole buffer.

3.27.2.12 vector< vector< int > > RoggenBuffer::getFrames (int *start*, int *end*)

This method returns a part of the buffer. The parameters tell from which start index to which stop index the frames should be returned. The frame at the start position is the first to be returend. And the frame at the end position is the last to be returend. Same functionality as getRange.

3.27.2.13 vector< vector< int > > RoggenBuffer::getLast ()

Getter function for the last element of the buffer. The return value has the same format as the buffer itself.

3.27.2.14 vector< vector< int > > * RoggenBuffer::getPointer ()

Getter function for the buffer pointer.

3.27.2.15 vector< vector< int > > RoggenBuffer::getRange (int *start*, int *end*)

Getter function for a certain part of the buffer. The parameters give start and end index.

3.27.2.16 vector< int > RoggenBuffer::getSingleMeasurement (int *meas*)

Returns a vector of a single measurement for all frames in the buffer. A measurement for examle is AccX. The measurement index must be within the valid range for the vector index.

3.27.2.17 bool RoggenBuffer::isFresh ()

Returns if there wasn't a read operation after the last write operation.

3.27.2.18 bool RoggenBuffer::isFull ()

Returns if the buffer is full. Full means there are as much frames as there is space in the buffer. There's nothing bad about this. When full the buffer works in a usual fifo manner (queue).

3.27.2.19 int RoggenBuffer::maxSize ()

Returns the current maximum length of the buffer.

3.27.2.20 void RoggenBuffer::resetUnlimited ()

Set the buffer to limited length. This means if full there is always the oldest frame removed when inserting a new one. A limited buffer is full when acting as a Queue.

3.27.2.21 void RoggenBuffer::setLength (int *bufferSize*)

Clears the internal buffer and sets its maximal length to the given length. The length can be larger then the actual number of frames in the buffer. In this case the buffer fills up before it acts like a queue.

3.27.2.22 void RoggenBuffer::setUnlimited (bool *flag*)

Sets unlimited mode of the buffer length. True means there are no more frames removed when inserting a new one. An unlimited buffer never gets full.

3.27.2.23 void RoggenBuffer::setUnlimited ()

Set the buffer to unlimited length. This means there are no more frames removed when inserting a new one. An unlimited buffer never gets full.

3.27.2.24 int RoggenBuffer::size ()

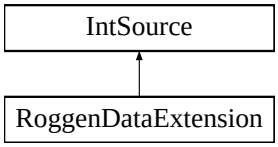
Returns the actual number of frames in the buffer.

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

- RoggenBuffer.h
- RoggenBuffer.cpp

3.28 RoggenDataExtension Class Reference

`#include <RoggenDataExtension.h>`Inheritance diagram for RoggenDataExtension::



Public Member Functions

- RoggenDataExtension (IntSource ∗)
- virtual ∼RoggenDataExtension ()
- virtual std::vector< std::vector< int > > getData ()

3.28.1 Detailed Description

Implements the same interface as the Sensor class itself. Therefore RoggenDataExtension and RoggenSensor can substitute themselfes. The idea of the extension is to compute additional values and append them to the original sensor measurements.

3.28.2 Constructor & Destructor Documentation

3.28.2.1 RoggenDataExtension::RoggenDataExtension (IntSource ∗ *givenSource*)

Constructor: The original source has to be given when constructing the extension. This class just adds two computed measurements to the stream.

3.28.2.2 RoggenDataExtension::∼RoggenDataExtension () `[virtual]`

Empty destructor.

3.28.3 Member Function Documentation

3.28.3.1 vector< vector< int > > RoggenDataExtension::getData () `[virtual]`

Overwrites the getData() function of the IntSource interface. The given data is used to compute two more measurements. The extra data then gets appended to the end of each frame.

Implements IntSource.

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

- RoggenDataExtension.h
- RoggenDataExtension.cpp

3.29 RoggenFeatureExtraction Class Reference

`#include <RoggenFeatureExtraction.h>`

Public Member Functions

- RoggenFeatureExtraction (RoggenBuffer *)
- virtual ~RoggenFeatureExtraction ()
- float getMean (int)
- float getVariance (int, float)
- float getVariance (int)
- float getStandardDeviation (int, float)
- float getStandardDeviation (int)
- float getMeanCrossingRate (int, float)
- float getMeanCrossingRate (int)
- float getMean (int, int, int)
- float getVariance (int, int, int, float)
- float getStandardDeviation (int, int, int, float)
- float getMeanCrossingRate (int, int, int, float)
- float getVariance (int, int, int)
- float getStandardDeviation (int, int, int)
- float getMeanCrossingRate (int, int, int)
- void **setIndices** (std::vector< int >)
- bool gotEnergy (int, int, int)
- bool lostEnergy (int, int, int)
- bool gotEnergy (int, int)
- bool lostEnergy (int, int)
- bool gotEnergy ()
- bool lostEnergy ()
- bool gotEnergy (int)
- bool lostEnergy (int)

Static Public Member Functions

- static float **getMean** (std::vector< int >)
- static float **getVariance** (std::vector< int >, float)
- static float **getStandardDeviation** (std::vector< int >, float)
- static float **getMeanCrossingRate** (std::vector< int >, float)
- static float **getVariance** (std::vector< int >)
- static float **getStandardDeviation** (std::vector< int >)
- static float **getMeanCrossingRate** (std::vector< int >)

3.29.1 Detailed Description

Class to handle the data processing of Daniel Roggen Sensors. The feature extraction works on a buffer. It offers easy access to many types of features. Energy based segmentation is supported as well. Author: Lars Widmer, www.lawi.ch

3.29.2 Constructor & Destructor Documentation

3.29.2.1 RoggenFeatureExtraction::RoggenFeatureExtraction (RoggenBuffer * *b*)

Constructor: FeatureExtraction objects only work on a buffer. Therefore a pointer to a buffer has to be handed over for construction. Initially the index vector get's cleared. The indices are used for energy based segementation.

3.29.2.2 RoggenFeatureExtraction::~RoggenFeatureExtraction () `[virtual]`

Destructor: Ready to clean up... yet nothing to do.

3.29.3 Member Function Documentation

3.29.3.1 float RoggenFeatureExtraction::getMean (int *meas*, int *start*, int *end*)

Returns the mean value of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the measurement index you choose which measurement to use for calculating the average. With the second and third parameter the buffer range to use can be defined.

3.29.3.2 float RoggenFeatureExtraction::getMean (int *meas*)

Returns the mean value of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the measurement index you choose which measurement to use for calculating the average.

3.29.3.3 float RoggenFeatureExtraction::getMeanCrossingRate (int *meas*, int *start*, int *end*)

Returns the mean crossing rate of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the second and third parameter the buffer range to use can be defined. For computing the mean crossing rate the mean value is needed. If it has already been computed, use the overloaded function and pass the mean as the last parameter for better performance.

3.29.3.4 float RoggenFeatureExtraction::getMeanCrossingRate (int *meas*, int *start*, int *end*, float *mean*)

Returns the mean crossing rate of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the second and third parameter the buffer range to use can be defined. For computing the mean crossing rate the mean value is needed. If it has already been computed, pass it as the last parameter for better performance. There's also a version of getMeanCrossingRate without the mean parameter.

3.29.3.5 float RoggenFeatureExtraction::getMeanCrossingRate (int *meas*)

Returns the mean crossing rate of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. For computing the mean crossing rate the mean value is needed. If it has already been computed, use the overloaded function and pass the mean as the second parameter for better performance.

3.29.3.6 float RoggenFeatureExtraction::getMeanCrossingRate (int *meas*, float *mean*)

Returns the mean crossing rate of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. For computing the mean crossing rate the mean value is needed. If it has already been computed, pass it as the second parameter for better performance. There's also a version of getMeanCrossingRate without a second parameter.

3.29.3.7 float RoggenFeatureExtraction::getStandardDeviation (int *meas*, int *start*, int *end*)

Returns the standard deviation of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the second and third parameter the buffer range to use can be defined. For computing the standard deviation the mean value is needed. If it has already been computed, use the overloaded function and pass the mean as the last parameter for better performance.

3.29.3.8 float RoggenFeatureExtraction::getStandardDeviation (int *meas*, int *start*, int *end*, float *mean*)

Returns the standard deviation of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the second and third parameter the buffer range to use can be defined. For computing the standard deviation the mean value is needed. If it has already been computed, pass it as the last parameter for better performance. There's also a version of getStandardDeviation without the mean parameter.

3.29.3.9 float RoggenFeatureExtraction::getStandardDeviation (int *meas*)

Returns the standard deviation of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. For computing the standard deviation the mean value is needed. If it has already been computed, use the overloaded function and pass the mean as the second parameter for better performance.

3.29.3.10 float RoggenFeatureExtraction::getStandardDeviation (int *meas*, float *mean*)

Returns the standard deviation of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. For computing the standard deviation the mean value is needed. If it has already been computed, pass it as the second parameter for better performance. There's also a version of getStandardDeviation without a second parameter.

3.29.3.11 float RoggenFeatureExtraction::getVariance (int *meas*, int *start*, int *end*)

Returns the variance of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the second and third parameter the buffer range to use can be defined. For computing the variance the mean value is needed. If it has already been computed, use the overloaded function and pass the mean as the last parameter for better performance.

3.29.3.12 float RoggenFeatureExtraction::getVariance (int *meas*, int *start*, int *end*, float *mean*)

Returns the variance of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the second and third parameter the buffer range to use can be defined. For computing the variance the mean value is needed. If it has already been computed, pass

it as the last parameter for better performance. There's also a version of getVariance without the mean parameter.

3.29.3.13 float RoggenFeatureExtraction::getVariance (int *meas*)

Returns the variance of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. For computing the variance the mean value is needed. If it has already been computed, use the overloaded function and pass the mean as the second parameter for better performance.

3.29.3.14 float RoggenFeatureExtraction::getVariance (int *meas*, float *mean*)

Returns the variance of the buffer for the given measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. For computing the variance the mean value is needed. If it has already been computed, pass it as the second parameter for better performance. There's also a version of getVariance without a second parameter.

3.29.3.15 bool RoggenFeatureExtraction::gotEnergy (int *numberOfFrames*)

Method used for energy based segementation. Returns true if the variance is above a fixed threshold plus hysteresis. For the measurement indices the indices stored by setIndices are used. For the range the given number of the last frames in the buffer is used.

3.29.3.16 bool RoggenFeatureExtraction::gotEnergy ()

Method used for energy based segementation. Returns true if the variance is above a fixed threshold plus hysteresis. For the range the whole buffer length is used. For the measurement indices the indices stored by setIndices are used.

3.29.3.17 bool RoggenFeatureExtraction::gotEnergy (int *start*, int *end*)

Method used for energy based segementation. Returns true if the variance is above a fixed threshold plus hysteresis. With the parameters the buffer range to use can be defined. For the measurement indices the indices stored by setIndices are used.

3.29.3.18 bool RoggenFeatureExtraction::gotEnergy (int *meas*, int *start*, int *end*)

Method used for energy based segementation. Returns true if the variance exceeds a fixed threshold plus hysteresis. This method just checks for a single measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the second and third parameter the buffer range to use can be defined.

3.29.3.19 bool RoggenFeatureExtraction::lostEnergy (int *numberOfFrames*)

Method used for energy based segementation. Returns true if the variance is below a fixed threshold minus hysteresis. For the measurement indices the indices stored by setIndices are used. For the range the given number of the last frames in the buffer is used.

3.29.3.20 bool RoggenFeatureExtraction::lostEnergy ()

Method used for energy based segementation. Returns true if the variance is below a fixed threshold minus hysteresis. For the range the whole buffer length is used. For the measurement indices the indices stored by setIndices are used.

3.29.3.21 bool RoggenFeatureExtraction::lostEnergy (int *start*, int *end*)

Method used for energy based segementation. Returns true if the variance is below a fixed threshold minus hysteresis. With the parameters the buffer range to use can be defined. For the measurement indices the indices stored by setIndices are used.

3.29.3.22 bool RoggenFeatureExtraction::lostEnergy (int *meas*, int *start*, int *end*)

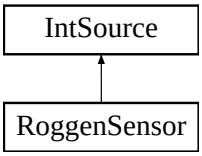
Method used for energy based segementation. Returns true if the variance is below a fixed threshold minus hysteresis. This method just checks for a single measurement. The buffer holds a set of measurements (AccX, AccY, RotX, ...) against time. With the second and third parameter the buffer range to use can be defined.

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

- RoggenFeatureExtraction.h
- RoggenFeatureExtraction.cpp

3.30 RoggenSensor Class Reference

Inheritance diagram for RoggenSensor::



Public Member Functions

- **RoggenSensor** (std::string, std::string)
- **RoggenSensor** (std::string)
- RoggenSensor (int)
- RoggenSensor ()
- virtual ~RoggenSensor ()
- virtual RoggenData getData ()

3.30.1 Constructor & Destructor Documentation

3.30.1.1 RoggenSensor::RoggenSensor (int *number*)

Constructor initializes the object with path (sensor device e.g. /dev/rfcomm0) but instead of the whole string just the number at the end is given. For the format the default value DX5;ccsss-s-s-ssssss is used.

3.30.1.2 RoggenSensor::RoggenSensor ()

Default constructor. Uses default path /dev/rfcomm0 For the format the default value DX5;ccsss-s-s-ssssss is used.

3.30.1.3 RoggenSensor::~RoggenSensor () `[virtual]`

Destructor: Closes the device and cleans up (deleting the frame parser).

3.30.2 Member Function Documentation

3.30.2.1 RoggenData RoggenSensor::getData () `[virtual]`

Implements the getData() method from the IntSource interface class. The method blocks when there is no data available. Therefore from the outside it looks and feels the same whether you use a single sensor (RoggenSensor) of a fusion (RoggenSensorFusion) of them. Both classes implement the same interface IntSource. The method calls the FrameParser from Daniel Roggen to convert the sensor data.

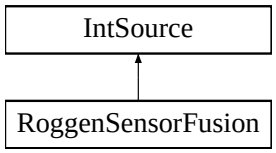
Implements IntSource.

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

- RoggenSensor.h
- RoggenSensor.cpp

3.31 RoggenSensorFusion Class Reference

`#include <RoggenSensorFusion.h>`Inheritance diagram for RoggenSensorFusion::



Public Member Functions

- **RoggenSensorFusion** (std::vector< std::string >, std::string)
- virtual ~RoggenSensorFusion ()
- virtual RoggenData getData ()
- void threadMethod ()

3.31.1 Detailed Description

This class mixes a number of sensors into a channel similar to a single sensor. Therefore it implements the same interface as the Sensor class itself. Still it's easy to see from which sensor a value originates since every frame carries the sensor ID within. Author: Lars Widmer, www.lawi.ch

3.31.2 Constructor & Destructor Documentation

3.31.2.1 RoggenSensorFusion::~RoggenSensorFusion () `[virtual]`

Destructor: Stops the threads and cleans up.

3.31.3 Member Function Documentation

3.31.3.1 RoggenData RoggenSensorFusion::getData () `[virtual]`

Implements the getData() method from the IntSource interface class. Reads data from the internal queue. Using conditional a blocking read is simulated. Therefore from the outside it looks and feels the same whether you use a single sensor (RoggenSensor) of a fusion (RoggenSensorFusion) of them. Both classes implement the same interface IntSource.

Implements IntSource.

3.31.3.2 void RoggenSensorFusion::threadMethod ()

Thread Method: This method is called once. After it runs out the thread stops. That's why it uses a while loop within. It get's data from the input sensors and puts it into a queue. The output method getData() reads from this queue. pthread conditionals are used to block getData() when there is no data available.

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

- RoggenSensorFusion.h

- RoggenSensorFusion.cpp

3.32 Sample Struct Reference

Public Attributes

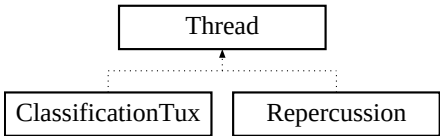
- Input **in**
- OutputType **out**

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

- LearningAlgorithm.h

3.33 Thread Class Reference

`#include <Thread.h>`Inheritance diagram for Thread::



Public Member Functions

- Thread ()
- Thread (bool)
- Thread (void(∗meth)(void))
- Thread (void(∗meth)(void), bool)
- virtual ∼Thread ()
- void setThreadMethod (void(∗meth)(void))
- void startThread ()
- void stopThread ()
- bool stopped ()
- virtual void threadMethod ()

3.33.1 Detailed Description

Author: Lars Widmer, www.lawi.ch

3.33.2 Constructor & Destructor Documentation

3.33.2.1 Thread::Thread ()

Constructor: Calls the init function to initialize the object. The thread isn't started by default.

3.33.2.2 Thread::Thread (bool *autostart*)

Constructor: If the parameter is set to true, the thread gets automatically started.

3.33.2.3 Thread::Thread (void(∗)(void) *meth*)

Constructor: The function in the parameter is used as thread function. This constructor could be helpful when aggregating this class.

3.33.2.4 Thread::Thread (void(∗)(void) *meth*, bool *autostart*)

Constructor: The function in the first parameter is used as thread function. If the second parameter is set to true, the thread gets automatically started. This constructor could be helpful when aggregating this class.

3.33.2.5 Thread::∼Thread () [virtual]

Destructor: Stops the thread and cleans up.

3.33.3 Member Function Documentation

3.33.3.1 void Thread::setThreadMethod (void(∗)(void) *meth*)

The function in the parameter is used as thread function. This constructor could be helpful when aggregating this class.

3.33.3.2 void Thread::startThread ()

Starts the thread.

3.33.3.3 bool Thread::stopped ()

Returns if the thread has been stopped. Use this method within the thread method. As soon as stopped gets true the thread method has to return.

3.33.3.4 void Thread::stopThread ()

Stops the thread and waits for its termination.

3.33.3.5 void Thread::threadMethod () [virtual]

By default this method calls the method set by setThreadMethod or the according constructors. If there hasn't been a method set a null pointer exception will occur. When inheriting this class this method should be overwritten! In general inheritance is the easier way of using this class. Aggregation is more useful when you want to have more then one thread.

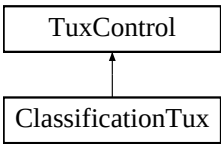
Reimplemented in ClassificationTux, and Repercussion.

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

- Thread.h
- Thread.cpp

3.34 TuxControl Class Reference

Inheritance diagram for TuxControl::



Public Member Functions

- TuxControl ()
- virtual ~TuxControl ()
- ControlEvent getControlEvent ()
- void setControlEvent (ControlEvent ce)
- void validateControlEvent ()
- void invalidateControlEvent ()
- ControlEvent removeControlEvent ()
- void addControlEvent (ControlEvent ce)
- int getEventQueueLength ()
- void clearQueue ()

3.34.1 Constructor & Destructor Documentation

3.34.1.1 TuxControl::TuxControl ()

Initializes the aggregated mutex. And initially clears the queue.

3.34.1.2 TuxControl::~TuxControl () `[virtual]`

Deletes the mutex.

3.34.2 Member Function Documentation

3.34.2.1 void TuxControl::addControlEvent (ControlEvent *ce*)

Adds a new element to the back of the queue.

3.34.2.2 void TuxControl::clearQueue ()

Clears the queue.

3.34.2.3 ControlEvent TuxControl::getControlEvent ()

Returns the first event in queue. It does neither invalidate nor remove the event. Use this function if you don't want queue functionality.

3.34.2.4 int TuxControl::getEventQueueLength ()

Returns the current length of the queue.

3.34.2.5 void TuxControl::invalidateControlEvent ()

Sets the first element in queue to be invalid. Use this function if you don't want queue functionality.

3.34.2.6 ControlEvent TuxControl::removeControlEvent ()

Removes and returns the first (oldest) element from the queue.

3.34.2.7 void TuxControl::setControlEvent (ControlEvent *ce*)

Clears the queue and sets the first event in queue. Use this function if you don't want queue functionality.

3.34.2.8 void TuxControl::validateControlEvent ()

Sets the first element in queue to be valid. Use this function if you don't want queue functionality.

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

- TuxControl.h
- TuxControl.cpp

3.35 TuxControlSingleton Class Reference

`#include <TuxControlSingleton.h>`

Static Public Member Functions

- static TuxControl * getInstance ()

3.35.1 Detailed Description

Singleton class limiting to one instance of TuxControl. Therefore the getInstance method always returns the pointer to the same instance of TuxControl. Of course singleton behaviour is only ensured if only this class gets used instead of TuxControl or it's descendant. Author: Lars Widmer, www.lawi.ch

3.35.2 Member Function Documentation

3.35.2.1 TuxControl * TuxControlSingleton::getInstance () [static]

Singleton constructor limiting to one instance of TuxControl. Therefore this method always returns the pointer to the same instance of TuxControl. Adapt the actual instatiation to use the required descendant of TuxControl (e.g. ClassificationTux).

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

- TuxControlSingleton.h
- TuxControlSingleton.cpp

3.36 XmlFileHandling Class Reference

Static Public Member Functions

- static void **storeNeighboursOld** (const Neighbours &neighbours, const std::string &filename)
- static Neighbours **readNeighboursOld** (const std::string &filename)
- static void **store** (const Neighbours &neighbours, const std::string &filename)
- static Neighbours **readNeighbours** (const std::string &filename)
- static void **store** (const FloatData &floatData, const std::string &filename)
- static FloatData **readFloatData** (const std::string &filename)
- static void **store** (const IntData &intData, const std::string &filename)
- static IntData **readIntData** (const std::string &filename)
- static FloatData **convert** (const Neighbours &neighbours, const std::string &name)
- static Neighbours **convert** (const FloatData &floatData)
- static FloatData **getEmptyFloatData** ()
- static FloatStruct **getEmptyFloatStruct** ()
- static IntData **getEmptyIntData** ()
- static IntStruct **getEmptyIntStruct** ()
- static void **print** (const FloatData &floatData)
- static void **print** (const Neighbours &neighbours)
- static void **print** (const IntData &intData)

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

- XmlFileHandling.h

Index