LibPololuRTDB

Generated by Doxygen 1.6.3

Tue Jul 12 15:13:48 2011

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 Angle Class Reference

Public Member Functions

- double GetPositive (void) const
- Angle (const int32_t degree)
- Angle (const double angle=0.0)
- Angle (const Angle &angle)
- double Get (void) const
- double toDouble (void) const
- double Abs (void) const
- int32_t Deg (void) const
- void Set (double angle)

2.1.1 Detailed Description

Class for handling angles The Angle Classes main job is to make sure that the angles are in the range of -M_PI to M_PI. This is the convention which is used in this API. Furthermore the class allows the comparisons of two angles. You can also do arithmetic operations with the angle objects. And finally the class makes it easy to convert between the radian and the degree representation of an angle and allows to use the stream output << with angles (they are output as "DEG°").

2.1.2 Constructor & Destructor Documentation

2.1.2.1 Angle::Angle (const int32_t *degree*) `[inline]`

Constructor for Angle class

Parameters

← ***degree*** Initial angle value in degree.

2.1.2.2 Angle::Angle (const double *angle* = 0.0) [inline]

Constructor for Angle class

Parameters

← ***angle*** Initial angle value in radian.

2.1.2.3 Angle::Angle (const Angle & *angle*) [inline]

Constructor for Angle class

Parameters

← ***angle*** Initial angle value from an other Angle object.

2.1.3 Member Function Documentation

2.1.3.1 double Angle::Abs (void) const [inline]

Get the absolut angle in radian.

Returns

Absolut angle in radian.

2.1.3.2 int32_t Angle::Deg (void) const [inline]

Get the angle in degree.

Returns

angle in degree.

2.1.3.3 double Angle::Get (void) const [inline]

Get the angle in radian.

Returns

angle in radian.

See also

Rad(void) const

2.1.3.4 double Angle::GetPositive (void) const [inline]

Rotates all angles in order to make them all positive. Used to enable comparisons between angles

2.1.3.5 void Angle::Set (double *angle*) [inline]

Set the angle.

Parameters

← ***angle*** New angle value in radian.

2.1.3.6 double Angle::toDouble (void) const [inline]

Get the angle in radian.

Returns

angle in radian.

See also

Get(void) const

2.2 Position Class Reference

Public Member Functions

- Position (double x=0.0, double y=0.0)
- double DistanceTo (const Position &dest) const
- Angle AngleOfLineToPos (const Position &dest)

2.2.1 Detailed Description

Class for handling positions Position class which can hold the position of a robot or any other position. The class also handles additional common computations with positions like computing the euclidian distance between two positions.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 Position::Position (double *x* = 0.0, double *y* = 0.0) [inline]

Initializes the Position to (0.0/0.0).

2.2.3 Member Function Documentation

2.2.3.1 Angle Position::AngleOfLineToPos (const Position & *dest*) [inline]

Computes the angle of a line between the objects position and the destination position.

Parameters

← ***dest*** Destination Position for the calcualation

Returns

Angle to this position. If dest == own Position the methode returns a zero Angle.

2.2.3.2 double Position::DistanceTo (const Position & *dest*) const [inline]

Computes the distance between the objects position and the one given by the argument.

Parameters

← ***dest*** The distance to this Position is calculated.

Returns

The computed distance.

2.3 RawBall Class Reference

Public Member Functions

- RawBall (RTDBConn &DBC)
- double GetX (void)
- double GetY (void)
- Angle GetPhi (void)
- double GetVelocity (void)
- Position GetPos (void)

Static Public Attributes

- static const double BALL_MOVING_TRESHOLD = 0.001

Protected Member Functions

- void UpdateBallInfoIfNecessary (void)

Protected Attributes

- Cam mCam
- BallInfo mBall

2.3.1 Detailed Description

Simple ball class which gives access to all ball informations stored in the RTDB. The RawBall class gives a simple access to the ball informations which are extracted by soccerlab_vision process and then stored in the RTDB. The informations about the ball are:

- Position on the playground.
- velocity in m/s
- the angle of the movement on the playground (phi)

2.3.2 Constructor & Destructor Documentation

2.3.2.1 RawBall::RawBall (RTDBConn & *DBC*) `[inline]`

Gets the first ballInfo at construction

2.3.3 Member Function Documentation

2.3.3.1 Angle RawBall::GetPhi (void) `[inline]`

Get the current moving direction of the ball.

Returns

Moving direction as an Angle object.

2.3.3.2 Position RawBall::GetPos (void) [inline]

Get the current position of the ball

Returns

Position of the ball as a Position object.

2.3.3.3 double RawBall::GetVelocity (void) [inline]

Get the current velocity of the ball.

Returns

Velocity in m/s.

2.3.3.4 double RawBall::GetX (void) [inline]

Get the current x-coordinate of the ball.

Returns

X-coordinate in m.

2.3.3.5 double RawBall::GetY (void) [inline]

Get the current y-coordinate of the ball.

Returns

Y-coordinate in m.

2.3.3.6 void RawBall::UpdateBallInfoIfNecessary (void) [inline, protected]

Checks whether new valid data is present and if so updates mBall.

2.3.4 Member Data Documentation

2.3.4.1 const double RawBall::BALL_MOVING_TRESHOLD = 0.001 [static]

If the velocity of the ball is beneath this threshold the ball is assumed to stand still.

2.3.4.2 BallInfo RawBall::mBall [protected]

Struct for the storage of all the information of the ball.

2.3.4.3 Cam RawBall::mCam [protected]

Cam object which allows to retrieve the informations of the ball from the camera.

2.4 Referee Class Reference

Public Member Functions

- Referee (class RTDBConn &DBC, const char *name="rtdb_referee", const int &otype=KOGMO_-RTDB_OBJTYPE_POLOLU, const int32_t &child_size=0, char **child_dataptr=NULL)
- void Init (void)
- ePlayMode GetPlayMode (void)
- int GetLeftSideGoals (void)
- int GetRightSideGoals (void)
- eSide GetSide (void)
- void SetReady (int side)
- void SetBlueReady (void)
- void SetRedReady (void)

Protected Attributes

- kogmo_rtdb_subobj_pololu_referee_t * obj_pololu_referee

2.4.1 Detailed Description

Referee class which handles the connection with the referee process via the RTDB. An object of this class must be polled continuously to obey the referees commands

If the referee is in init state GetPlayMode() will return REFEREE_INIT If the person who control the referee decides which team has to play on the left side, the state changes to BEFORE_KICK_OFF.

At this time you can receive GetBlueSide(). The result is either LEFT_SIDE or RIGHT_SIDE You need to know your own team so that you can make use of this information, e.g. you are team red and receive that blue plays on the left side => you play on right side.

If you want to know which team may do the kick of then use GetSide(). It returns LEFT_SIDE or RIGHT_-SIDE and you have to logically think what that means for your team.

Now you should drive to kick-off positions and then do SetReady(int team) including your team (Blue=0, Red=1)

If there is any change of sides or another kick-off, the referee will change his state to BEFORE_KICK_OFF and you can then get the information by GetBlueSide() and GetSide().

While playing the referee will stay in state PLAY_ON. If the time is over you will receive the state TIME_-OVER and a following BEFORE_KICK_OFF

2.4.2 Constructor & Destructor Documentation

2.4.2.1 Referee::Referee (class RTDBConn & *DBC*, const char * *name* = `"rtdb_referee"`, const int & *otype* = `KOGMO_RTDB_OBJTYPE_POLOLU`, const int32_t & *child_size* = `0`, char ** *child_dataptr* = `NULL`)

2.4.3 Member Function Documentation

2.4.3.1 int Referee::GetLeftSideGoals (void) `[inline]`

Retrieves the current score of the left side.

2.4.3.2 ePlayMode Referee::GetPlayMode (void) [inline]

Retrieves the current play mode from the referee. for more help concerning the PlayModes look up the enum ePlayMode in share.h

2.4.3.3 int Referee::GetRightSideGoals (void) [inline]

Retrieves the current score of the right side.

2.4.3.4 eSide Referee::GetSide (void) [inline]

Retrieves the current value of the side. This is used to find out which side has kick-off or penalty in the following way:

Returns

0 means left, 1 means right

ePlayMode = BLUE_LEFT or RED_LEFT and ePlayMode = KICK_OFF: GetSide() returns the side of the Team which has kick-off.

in all other ePlayMode's the return of GetSide is not defined (and does not matter)

2.4.3.5 void Referee::Init (void)

Initializes the referee status.

2.4.3.6 void Referee::SetBlueReady (void) [inline]

Here you can set ready the blue team

2.4.3.7 void Referee::SetReady (int *side*) [inline]

Here you can set if you are ready.

Parameters

← ***side*** Has to be 0 if you are blue, 1 if you are red.

2.4.3.8 void Referee::SetRedReady (void) [inline]

Here you can set ready the red team

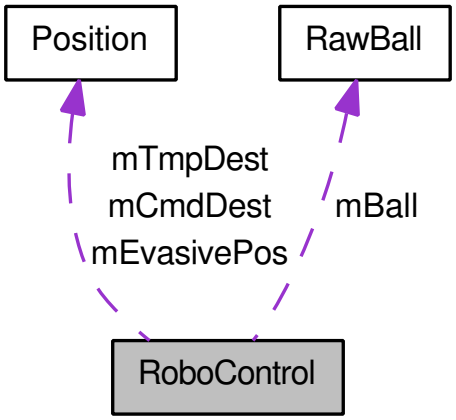
2.4.4 Member Data Documentation

2.4.4.1 kogmo_rtdb_subobj_pololu_referee_t* Referee::obj_pololu_referee [protected]

Struct which holds all information that is stored in the RDTB.

2.5 RoboControl Class Reference

Collaboration diagram for RoboControl:



Public Member Functions

- RoboControl (RTDBConn &DBC, const int deviceNr)
- bool Kick (eForce force, double distToBall=0.0)
- bool Kick (uint32_t force, double distToBall=0.0)
- void UpdateTeamInfoIfNecessary (TeamInfo &team_info, const eTeam team)
- void GetMac (uint8_t(&mac)[6]) const
- uint8_t GetDistanceSensors (void) const
- uint32_t GetAccuVoltage (void) const
- eMovingStatus GetMovingStatus (void) const
- eLocalizationStatus GetLocalizationStatus (void) const
- eModuleStatus GetModuleStatus (void) const
- int32_t GetSpeedLeft (void) const
- int32_t GetSpeedRight (void)
- uint32_t GetMsToGo (void)
- double GetX (void)
- double GetY (void)
- Angle GetPhi (void)
- Position GetPos (void)
- int GetRfcommNr (void)
- void MoveMs (int32_t left, int32_t right, uint32_t ms, uint32_t rampUp=DEFAULT_RAMPUP_TIME)
- void MoveMsBlocking (int32_t left, int32_t right, uint32_t ms, uint32_t rampUp=DEFAULT_RAMPUP_TIME)
- void Turn (const Angle &angle)
- void TurnAbs (const Angle &angle)
- void GotoXY (const double x, const double y, int32_t speed=160, const bool precise=true)
- void GotoPos (Position dest)
- void StopAction (void)
- void TurnBlocking (const Angle &angle)
- void MoveDist (double dist, int32_t speed=160, bool precise=true)

2.5.1 Detailed Description

Class for the basic control over the robots. The RoboControl class gives you access to all functionality of the robot on the playground. This means on the one hand you can get all status information about the robot which is controlled by the specific instance of the class. On the other hand you can issue all possible commands via this class. The class uses the PololuCmd and the PololuStatus to communicate with the pololu_rtdb_module (and thus the robot) via the RTDB. It is not recommended to use the PololuCmd or the PololuStatus classes directly!

2.5.2 Constructor & Destructor Documentation

2.5.2.1 RoboControl::RoboControl (RTDBConn & *DBC*, const int *deviceNr*)

Constructor gets first robo positions.

2.5.3 Member Function Documentation

2.5.3.1 uint32_t RoboControl::GetAccuVoltage (void) const `[inline]`

Get the accuvoltage of the robot.

Returns

Accuvoltage in mV.

2.5.3.2 uint8_t RoboControl::GetDistanceSensors (void) const `[inline]`

Get the values of the distance sensors.

2.5.3.3 eLocalizationStatus RoboControl::GetLocalizationStatus (void) const `[inline]`

Get the status of the localization procedure. Be careful, that the pololu_rtdb_module might think that it is located properly but in fact it is not. The information STATUS_INT_LOCALIZED does only mean, that the localization is finished but not that the robot was localized correctly (how would he know?!?) however a wrong localization should be very rare... The only error which can occur here is that the localization of robots in a team could be interchanged or that they are equal.

Returns

Localization status in form of an enum eLocalizationStatus

2.5.3.4 void RoboControl::GetMac (uint8_t(&) *mac*[6]) const `[inline]`

Get the mac address of the robot.

Parameters

→ ***mac*** Mac as an array of six unsigned characters.

2.5.3.5 eModuleStatus RoboControl::GetModuleStatus (void) const [inline]

Get the module status. This means get the status of either the gotoXY command or the MoveVector command. This is helpful to determine if one of these commands is already finished.

Returns

Modul status (pololu_rtdb_modul) in form of an enum eModuleStatus

2.5.3.6 eMovingStatus RoboControl::GetMovingStatus (void) const [inline]

Get the moving status of the robot. This means: is the robot moving or turning or standing... Be careful this information is updated AFTER the robot really is in this moving state. This can lead to problems when you issue for example a turning command because the confirm of the robot might take some ms!

Returns

Movingstatus in form of an enum eMovingStatus

2.5.3.7 uint32_t RoboControl::GetMsToGo (void) [inline]

Get the ms to move from the status of the robot.

2.5.3.8 Angle RoboControl::GetPhi (void) [inline]

Get the absolute orientation of the robot.

2.5.3.9 Position RoboControl::GetPos (void) [inline]

Gets the cartesian coordinates of the robot.

2.5.3.10 int RoboControl::GetRfcommNr (void) [inline]

Gets the Rfcomm Number which the robot is connected to.

2.5.3.11 int32_t RoboControl::GetSpeedLeft (void) const [inline]

Get the left speed from the status of the robot.

2.5.3.12 int32_t RoboControl::GetSpeedRight (void) [inline]

Get the right speed from the status of the robot.

2.5.3.13 double RoboControl::GetX (void) [inline]

Get the x-coordinate of the robot.

2.5.3.14 double RoboControl::GetY (void) [inline]

Get the y-coordinate of the robot.

2.5.3.15 void RoboControl::GotoPos (Position *dest*) [inline]

Tell the pololu_rtdb_module to move the robot to a specific place. Uses GotoXY internally

Parameters

← ***dest*** destination in form of a Position object.

2.5.3.16 void RoboControl::GotoXY (const double *x*, const double *y*, int32_t *speed* = 160, const bool *precise* = true) [inline]

Tell the pololu_rtdb_module to move the robot to a specific place.

Warning

Do not change the default speed!

Parameters

← ***x*** x-coordinate of the goal position.

← ***y*** y-coordinate of the goal position.

← ***speed*** Speed with which to move.

← ***precise*** Toggles if the robot runs precise at the end of a route (rampDown on if precise=true)

2.5.3.17 bool RoboControl::Kick (uint32_t *force*, double *distToBall* = 0.0)

Kick methode

Returns

true if kick has been executed and false if not

Parameters

← ***force*** Force of the kick: integer value between 0 and 100 where 0 is equivalent to SOFT and 100 equivalent to HARD

→ ***distToBall*** Distance which the Robot will move forward in order to kick If the default = 0.0 is specified the method will measure the distance to the ball and move this distance. Distances bigger then 25cm are not allowed.

2.5.3.18 bool RoboControl::Kick (eForce *force*, double *distToBall* = 0.0)

Path planning algorithm to move to a certain destination (dest) which avoids other players. This method has to be called in a loop in order to work. It does not block the program execution.

Returns

ePathMovingStatus: Gives you a clue on what the satus of the path planning algorithm is like.

Parameters

← ***pathLength*** The maximum length of the single straight path which in their sum make the complex path to the destination.

← ***dest*** The destination which the robot should be moved to. Kick methode

Returns

true if kick has been executed and false if not

Parameters

← ***force*** Force of the kick: SOFT MEDIUM HARD

→ ***distToBall*** Distance which the Robot will move forward in order to kick If the default = 0.0 is specified the methode will measure the distance to the ball and move this distance. Distances bigger then 25cm are not allowed.

2.5.3.19 void RoboControl::MoveDist (double *dist*, int32_t *speed* = `160`, bool *precise* = `true`) `[inline]`

Move forward and stop after a specified distance. It is possible to define speed and precise value.

Parameters

← ***dist*** Distance which should be moved. It is not possible to drive backward

2.5.3.20 void RoboControl::MoveMs (int32_t *left*, int32_t *right*, uint32_t *ms*, uint32_t *rampUp* = `DEFAULT_RAMPUP_TIME`) `[inline]`

Set the motor speeds to specific values with a timeout. A rampuptime can be specified. This is recommended when you want to drive faster than 120 otherwise the motors might slip... The default rampuptime is 200ms.

Parameters

← ***left*** left motor speed as a raw value ranging from -255 to 255.

← ***right*** right motor speed as a raw value ranging from -255 to 255.

← ***ms*** ms which the movement should last.

← ***rampUp*** ramup time. The motors will get faster in a linear way during this period of time.

2.5.3.21 void RoboControl::MoveMsBlocking (int32_t *left*, int32_t *right*, uint32_t *ms*, uint32_t *rampUp* = `DEFAULT_RAMPUP_TIME`)

Same as MoveMs with the difference that the program execution will be blocked until either the robot will confirm that the movement has stopped or a timeout has expired (timeout is calculated to be 400ms more than the anticipated movement time)

2.5.3.22 void RoboControl::StopAction (void) [inline]

Issues the pololu_rtdb_module to (re)localize the robot. This is useful if for example you have two robots with the same position...

Returns

true if robot is now localized and false if not. Stops any movement

2.5.3.23 void RoboControl::Turn (const Angle & *angle*) [inline]

Turn a specified angle. Clock-wise is positive and counter-clock-wise is negative.

Parameters

← ***angle*** angle which should be rotated as an Angle object. To turn via a given value in degree or double just use Angle(degree/double)

2.5.3.24 void RoboControl::TurnAbs (const Angle & *angle*) [inline]

Turn to an absolute angle

Parameters

← ***angle*** to turn to

2.5.3.25 void RoboControl::TurnBlocking (const Angle & *angle*)

Same as Turn with the difference that the program execution will be blocked until either the robot will confirm that the rotation has stopped or a timeout has expired (timeout is calculated to be more than the anticipated movement time)

Parameters

← ***angle*** angle which should be rotated as an Angle object.

2.5.3.26 void RoboControl::UpdateTeamInfoIfNecessary (TeamInfo & *team_info*, const eTeam *team*) [inline]

Update the TeamInfo only if necessary. You have to use this method to keep the TeamInfo structs up-to-date

Index