DeltaRobot

v0.4

Generated by Doxygen 1.8.9.1

Tue May 26 2015 14:49:15

Contents

Chapter 1

Main Page

This project is a Delta robot controller using Dynamixel AX12 servos.This type of robot can pick and place objects

Chapter 2

Namespace Documentation

2.1 Ui Namespace Reference

Namespace to work with a User Interface Qt Form.

2.1.1 Detailed Description

Namespace to work with a User Interface Qt Form.

Chapter 3

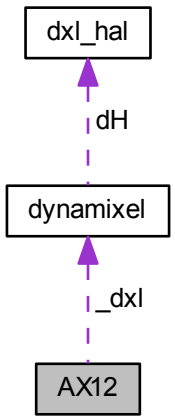
Class Documentation

3.1 AX12 Class Reference

The AX12 class is used to control AX-12 motors from Dynamixel.

`#include <ax12.h>`

Collaboration diagram for AX12:



Public Member Functions

- AX12 ()
  Default constructor.
- AX12 (dynamixel *_dxl, int ID=-1)
  Initializator constructor if ID == -1 no action is done.
- AX12 (const AX12 &a)
  Copy constructor.
- ~AX12 ()
  Default destructor.
- QVector< int > connectedID ()

Returns all active servos;.

- double getCurrentLoad ()

  Returns the current load from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.

- double getCurrentPos ()

  Returns the current position from 0º to 300º

- int getCurrentTemp ()

  Returns the current Temperature in Celsius.

- double getCurrentSpeed ()

  Returns the current speed from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.

- double getCurrentVoltage ()

  Returns the current voltage in Volts.

- int getID ()

  To get the current ID.

- void setComplianceSlope (uchar ccw, uchar cw)

  Sets the compliance slope.

- void setDxl (dynamixel *dxl)

  Sets the dynamixel interface.

- void setGoalPosition (double goal)

  Sets the Goal's position (in degrees) or speed depending on the mode.

- void setID (int ID)

  To set a new ID.

- void setJointMode (bool mode)

  To set Joint/Wheel mode.

- void setMinMax (double min, double max)

  To set the minimum and maximum angle from 0 to 300º

- void setRadians (bool rads)

  Sets the radians mode.

- void setSpeed (double speed)

  To set the maximum speed from 0% to 100% if joint mode or from -100% to 100% if wheel mode.

Private Types

- enum ROM {
  ModelNumber = 0, VersionFirmware = 2, ID = 3, BaudRate = 4,
  ReturnDelayTime = 5, CWAngleLimit = 6, CCWAngleLimit = 8, HighestLimitTemp = 11,
  LowestLimitVoltage = 12, HighestLimitVoltage = 13, MaxTorque = 14, StatusReturnLevel = 16,
  AlarmLED = 17, AlarmShutdown = 18 }

  Contains all the EEPROM directions enumeration.

- enum RAM {
  TorqueEnable = 24, LED = 25, CWComplianceMargin = 26, CCWComplianceMargin = 27,
  CWComplianceSlope = 28, CCWComplianceSlope = 29, GoalPosition = 30, MovingSpeed = 32,
  TorqueLimit = 34, PresentPosition = 36, PresentSpeed = 38, PresentLoad = 40,
  PresentVoltage = 42, PresentTemperature = 43, Registered = 44, Moving = 46,
  Lock = 47, Punch = 48 }

  Contains all the RAM directions enumerations.

Private Attributes

- dynamixel * _dxl

  Contains the dynamixel comunication.
- int _ID

  Stores the current ID.
- bool _mode

  True if we use the joint mode.
- bool _rads

  True if the angle is returned in radians.

3.1.1 Detailed Description

The AX12 class is used to control AX-12 motors from Dynamixel.

3.1.2 Member Enumeration Documentation

3.1.2.1 enum AX12::RAM [private]

Contains all the RAM directions enumerations.

Enumerator

TorqueEnable

LED

CWComplianceMargin

CCWComplianceMargin

CWComplianceSlope

CCWComplianceSlope

GoalPosition

MovingSpeed

TorqueLimit

PresentPosition

PresentSpeed

PresentLoad

PresentVoltage

PresentTemperature

Registered

Moving

Lock

Punch

```
00044     {
00045         TorqueEnable        = 24,
00046         LED                 = 25,
00047         CWComplianceMargin  = 26,
00048         CCWComplianceMargin = 27,
00049         CWComplianceSlope   = 28,
00050         CCWComplianceSlope  = 29,
00051         GoalPosition        = 30,
00052         MovingSpeed         = 32,
00053         TorqueLimit         = 34,
00054         PresentPosition     = 36,
00055         PresentSpeed        = 38,
00056         PresentLoad         = 40,
```

```
00057         PresentVoltage      = 42,
00058         PresentTemperature  = 43,
00059         Registered          = 44,
00060         Moving              = 46,
00061         Lock                = 47,
00062         Punch               = 48
00063
00064     };
```

3.1.2.2 enum AX12::ROM [private]

Contains all the EEPROM directions enumeration.

Enumerator

ModelNumber

VersionFirmware

ID

BaudRate

ReturnDelayTime

CWAngleLimit

CCWAngleLimit

HighestLimitTemp

LowestLimitVoltage

HighestLimitVoltage

MaxTorque

StatusReturnLevel

AlarmLED

AlarmShutdown

```
00025     {
00026         ModelNumber         = 0,
00027         VersionFirmware     = 2,
00028         ID                  = 3,
00029         BaudRate            = 4,
00030         ReturnDelayTime     = 5,
00031         CWAngleLimit        = 6,
00032         CCWAngleLimit       = 8,
00033         HighestLimitTemp    = 11,
00034         LowestLimitVoltage  = 12,
00035         HighestLimitVoltage = 13,
00036         MaxTorque           = 14,
00037         StatusReturnLevel   = 16,
00038         AlarmLED            = 17,
00039         AlarmShutdown       = 18
00040     };
```

3.1.3 Constructor & Destructor Documentation

3.1.3.1 AX12::AX12 ( )

Default constructor.

```
00005             :
00006     _dxl(NULL),
00007     _ID(-1),
00008     _mode(true),
00009     _rads(false)
00010 {
00011
00012 }
```

3.1.3.2 AX12::AX12 (dynamixel * _dxl, int ID = -1)

Initializator constructor if ID == -1 no action is done.

```
00014                                   :
00015     _dxl(dxl),
00016     _ID(ID),
00017    _mode(true),
00018    _rads(false)
00019 {
00020     if (_ID < 0 or _dxl == NULL) return;
00021     dxl->write_byte(_ID, RAM::TorqueEnable, true);
00022 }
```

3.1.3.3 AX12::AX12 (const AX12 & a)

Copy constructor.

```
00024                           :
00025     _dxl(a._dxl),
00026     _ID(a._ID),
00027     _mode(a._mode),
00028     _rads(a._rads)
00029 {
00030
00031 }
```

3.1.3.4 AX12::~AX12 ( )

Default destructor.

```
00034 {
00035
00036 }
```

3.1.4 Member Function Documentation

3.1.4.1 QVector< int > AX12::connectedID ( )

Returns all active servos;.

```
00039 {
00040     if (_dxl == NULL) return QVector<int> (0);
00041
00042     QVector <int> res;
00043     for (int i = 0; i < 256; ++i) {
00044         _dxl->ping(i);
00045         if (_dxl->get_comm_result() == COMM_RXSUCCESS) res.push_back(i);
00046     }
00047
00048     return res;
00049 }
```

3.1.4.2 double AX12::getCurrentLoad ( )

Returns the current load from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.

```
00052 {
00053     if (_ID < 0 or _dxl == NULL) return 0;
00054     int load = _dxl->read_word(_ID, RAM::PresentLoad);
00055     if (load >= 1024) load -= 1024;
00056     return double((load/1023)*100);
00057 }
```

3.1.4.3 double AX12::getCurrentPos ( )

Returns the current position from 0º to 300º

```
00060 {
00061     if (_ID < 0 or _dxl == NULL) return 0;
00062     int pos = _dxl->read_word(_ID, RAM::PresentPosition);
00063     if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00064
00065     if (_rads) return double((pos/1023.0)*(5.0*M_PI)/3.0);
00066     return double((pos/1023.0)*300);
00067 }
```

3.1.4.4 double AX12::getCurrentSpeed ( )

Returns the current speed from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.

```
00078 {
00079     if (_ID < 0 or _dxl == NULL) return 0;
00080     int speed = _dxl->read_word(_ID, RAM::PresentSpeed);
00081     if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00082     speed -= 1024;
00083     if (speed == -1024) speed = 0;
00084     return double((speed/1023.0)*100);
00085 }
```

3.1.4.5 int AX12::getCurrentTemp ( )

Returns the current Temperature in Celsius.

```
00070 {
00071     if (_ID < 0 or _dxl == NULL) return 0;
00072     int temp = _dxl->read_byte(_ID, RAM::PresentTemperature);
00073     if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00074     return temp;
00075 }
```

3.1.4.6 double AX12::getCurrentVoltage ( )

Returns the current voltage in Volts.

```
00088 {
00089     if (_ID < 0 or _dxl == NULL) return 0;
00090     char voltage = _dxl->read_byte(_ID, RAM::PresentVoltage);
00091     if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00092     return double(voltage/10.0);
00093 }
```

3.1.4.7 int AX12::getID ( ) [inline]

To get the current ID.

```
00114 { return _ID; }
```

3.1.4.8 void AX12::setComplianceSlope (uchar *ccw*, uchar *cw*)

Sets the compliance slope.

Parameters

ccw	Counter Clock Wise Compliance Slope
cw	Clock Wise Compliance Slope

```
00096 {
00097     if (_ID < 0 or _dxl == NULL) return;
00098     _dxl->write_byte(_ID, RAM::CCWComplianceMargin, ccw);
00099     _dxl->write_byte(_ID, RAM::CWComplianceMargin, cw);
00100 }
```

3.1.4.9 void AX12::setDxl (dynamixel * *dxl*) [inline]

Sets the dynamixel interface.

Parameters

dxl	Pointer to the dynamixel control class

```
00123 { _dxl = dxl; }
```

3.1.4.10 void AX12::setGoalPosition (double *goal*)

Sets the Goal's position (in degrees) or speed depending on the mode.

Parameters

goal	Position (in degrees if not radian mode) or % speed if used wheel mode

```
00103 {
00104     if (_ID < 0 or _dxl == NULL) return;
00105
00106     // Conversion to radians if radians mode
00107     if (_rads) goal *= 180/M_PI;
00108
00109     if (goal > 300.0) goal = 300.0;
00110     else if (goal < 0) goal = 0;
00111     _dxl->write_word(_ID, RAM::GoalPosition, int((goal/300.0)*1023));
00112 }
```

3.1.4.11 void AX12::setID (int *ID*)

To set a new ID.

Parameters

ID	the new ID

```
00115 {
00116     _ID = ID;
00117     if (_ID < 0 or _dxl == NULL) return;
00118     _dxl->write_byte(_ID, RAM::TorqueEnable, true);
00119 }
```

3.1.4.12 void AX12::setJointMode (bool *mode*)

To set Joint/Wheel mode.

Parameters

mode	True if Joint and false if Wheel mode

```
00122 {
00123     if (_ID < 0 or _dxl == NULL) return;
00124     _mode = mode;
00125     if (_mode) {
00126         _dxl->write_word(_ID, ROM::CWAngleLimit, 0);
00127         _dxl->write_word(_ID, ROM::CCWAngleLimit, 1023);
00128     }
00129     else {
00130         _dxl->write_word(_ID, ROM::CWAngleLimit, 0);
00131         _dxl->write_word(_ID, ROM::CCWAngleLimit, 0);
00132     }
00133 }
```

3.1.4.13 void AX12::setMinMax (double *min*, double *max*)

To set the minimum and maximum angle from 0 to 300º

Parameters

min	Minimum value from servo
max	Maximum value from servo

```
00136 {
00137     if (_ID < 0 or _dxl == NULL) return;
00138
00139     if (min > max)  {
00140         double aux = min;
00141         min = max;
00142         max = aux;
00143     }
00144
00145     if (_rads) min *= 180/M_PI;
00146
00147     if (min < 0.0) min = 0;
00148     if (max > 300.0) max = 300;
00149
00150     min = (min/300)*1023;
00151     max = (max/300)*1023;
00152
00153     _dxl->write_word(_ID, ROM::CWAngleLimit, int (min));
00154     _dxl->write_word(_ID, ROM::CCWAngleLimit, int (max));
00155 }
```

3.1.4.14 void AX12::setRadians (bool *rads*) [inline]

Sets the radians mode.

Parameters

rads	True if radians mode is used

```
00145 { _rads = rads; }
```

3.1.4.15 void AX12::setSpeed (double *speed*)

To set the maximum speed from 0% to 100% if joint mode or from -100% to 100% if wheel mode.

```
00158 {
00159     if (_ID < 0 or _dxl == NULL) return;
00160     if (speed > 100.0) speed = 100.0;
00161     if (_mode) {
00162         if (speed < 0.0) speed = 0.0;
00163
```

```
00164         int byte = int((speed/100.0) * 1024.0);
00165         if (speed == 100.0) byte = 0;
00166         _dxl->write_byte(_ID, RAM::MovingSpeed, byte);
00167     }
00168     else {
00169         if (speed < -100.0) speed = -100.0;
00170
00171         int byte = int(((speed + 100)/100.0) * 1024);
00172         _dxl->write_byte(_ID, RAM::MovingSpeed, byte);
00173     }
00174
00175 }
```

3.1.5 Member Data Documentation

3.1.5.1 dynamixel* AX12::_dxl [private]

Contains the dynamixel comunication.

3.1.5.2 int AX12::_ID [private]

Stores the current ID.

3.1.5.3 bool AX12::_mode [private]

True if we use the joint mode.

3.1.5.4 bool AX12::_rads [private]

True if the angle is returned in radians.

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

- dxl/ax12.h
- dxl/ax12.cpp

3.2 ServoThread::Dominoe Struct Reference

Struct to handle the dominoe pieces.

Public Member Functions

- bool operator< (const Dominoe &d) const

  Overloaded operator for comparisions.
- Dominoe & operator= (const Dominoe &d)

  Overloaded operator to copy.
- Dominoe ()

  Default constructor.
- Dominoe (double X, double Y, double ori)

  Initialization constructor.
- Dominoe (QVector2D point, double ori)

  Initialization constructor with vector.

Public Attributes

- double X

  X position.
- double Y

  Y position.
- double ori

  Orientation from X = 0 in degrees.

3.2.1 Detailed Description

Struct to handle the dominoe pieces.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ServoThread::Dominoe::Dominoe ( ) [inline]

Default constructor.

```
00061 : X(0), Y(0), ori(0) {}
```

3.2.2.2 ServoThread::Dominoe::Dominoe (double *X*, double *Y*, double *ori*) [inline]

Initialization constructor.

```
00064 : X(X), Y(Y), ori(ori) {}
```

3.2.2.3 ServoThread::Dominoe::Dominoe (QVector2D *point*, double *ori*) [inline]

Initialization constructor with vector.

```
00067 : X(point.x()), Y(point.y()), ori(ori) {}
```

3.2.3 Member Function Documentation

3.2.3.1 bool ServoThread::Dominoe::operator< (const Dominoe & *d*) const [inline]

Overloaded operator for comparisions.

```
00046         {
00047             if (this->X != d.X) return this->X < d.X;
00048             return this->Y < d.Y;
00049         }
```

3.2.3.2 Dominoe& ServoThread::Dominoe::operator= (const Dominoe & *d*) [inline]

Overloaded operator to copy.

```
00053         {
00054             this->X = d.X;
00055             this->Y = d.Y;
00056             this->ori = d.ori;
00057             return *this;
00058         }
```

3.2.4 Member Data Documentation

3.2.4.1 double ServoThread::Dominoe::ori

Orientation from X = 0 in degrees.

3.2.4.2 double ServoThread::Dominoe::X

X position.

3.2.4.3 double ServoThread::Dominoe::Y

Y position.

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

- servothread.h

3.3 dxl_hal Class Reference

Dynamixel SDK platform dependent.

`#include <dxl_hal.h>`

Public Member Functions

- bool open (QString &devName, int baudrate)
- void close (void)
- void clear (void)
- int change_baudrate (float baudrate)
- int write (unsigned char *pPacket, int numPacket)
- int read (unsigned char *pPacket, int numPacket)
- double get_curr_time ()
- bool isOpen ()

Private Attributes

- QSerialPort _serial
- int _time = 30
- bool _timed = false
- bool _open = false

3.3.1 Detailed Description

Dynamixel SDK platform dependent.

3.3.2 Member Function Documentation

3.3.2.1 int dxl_hal::change_baudrate (float *baudrate*)

```
00041 {
00042     bool res = _serial.setBaudRate(qint32(baudrate));
```

```
00043     return int(res);
00044
00045 }
```

3.3.2.2 void dxl_hal::clear (void)

```
00032 {
00033     // Clear communication buffer
00034
00035     if (!_serial.isOpen()) return;
00036     _serial.clear();
00037
00038 }
```

3.3.2.3 void dxl_hal::close (void)

```
00025 {
00026     // Closing device
00027     _serial.close();
00028     _open = false;
00029 }
```

3.3.2.4 double dxl_hal::get_curr_time ( )

```
00082 {
00083     return (double)QTime::currentTime().msecsSinceStartOfDay();
00084 }
```

3.3.2.5 bool dxl_hal::isOpen ( ) [inline]

```
00030 { return _open; }
```

3.3.2.6 bool dxl_hal::open (QString & *devName*, int *baudrate*)

```
00007 {
00008     // Opening device
00009     // devIndex: Device index
00010     // baudrate: Real baudrate (ex> 115200, 57600, 38400...)
00011     // Return: 0(Failed), 1(Succeed)
00012
00013     _serial.setPortName(devName);
00014     _serial.setBaudRate(qint32(baudrate));
00015     _serial.setDataBits(QSerialPort::Data8);
00016     _serial.setParity(QSerialPort::NoParity);
00017     _serial.setStopBits(QSerialPort::OneStop);
00018     _serial.setFlowControl(QSerialPort::NoFlowControl);
00019     if(not _serial.open(QIODevice::ReadWrite)) return false;
00020     _open = true;
00021     return true;
00022 }
```

3.3.2.7 int dxl_hal::read (unsigned char * *pPacket*, int *numPacket*)

```
00065 {
00066     // Recieving date
00067     // *pPacket: data array pointer
00068     // numPacket: number of data array
00069     // Return: number of data recieved. -1 is error.
00070     _timed = false;
00071     if (_serial.isOpen()) {
00072         int n = _serial.read((char*)pPacket, numPacket);
00073         _timed = _serial.waitForReadyRead(_time);
00074         _timed = not _timed;
00075         return n;
```

```
00076     }
00077     else return -1;
00078
00079 }
```

3.3.2.8 int dxl_hal::write (unsigned char * *pPacket*, int *numPacket*)

```
00048 {
00049     // Transmiting date
00050     // *pPacket: data array pointer
00051     // numPacket: number of data array
00052     // Return: number of data transmitted. -1 is error.
00053     _timed = false;
00054     if (_serial.isOpen()) {
00055         int n = _serial.write((char*)pPacket, numPacket);
00056         _timed = _serial.waitForBytesWritten(_time);
00057         _timed = not _timed;
00058         return n;
00059     }
00060     else return -1;
00061
00062 }
```

3.3.3 Member Data Documentation

3.3.3.1 bool dxl_hal::_open = false [private]

3.3.3.2 QSerialPort dxl_hal::_serial [private]

3.3.3.3 int dxl_hal::_time = 30 [private]

3.3.3.4 bool dxl_hal::_timed = false [private]

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

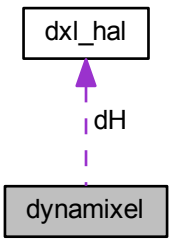
- dxl/dxl_hal.h
- dxl/dxl_hal.cpp

3.4 dynamixel Class Reference

Dynamixel 1.0 protocol class.

`#include <dynamixel.h>`

Collaboration diagram for dynamixel:



Public Member Functions

- dynamixel ()

  Default constructor.
- dynamixel (QString port_num, int baud_rate=1000000)

  Initialization constructor.
- ~dynamixel ()

  Default destructor.
- bool isOpen ()

  True if the port is open.
- int initialize (QString port_num, int baud_rate)

  Initializates the port.
- int change_baudrate (int baud_rate)

  Changes the current baud rate.
- int terminate (void)

  Closes the comunication.
- int get_comm_result ()

  Returns the current com status.
- void tx_packet (void)

  Sends a packet.
- void rx_packet (void)

  Receives a packet.
- void txrx_packet (void)

  Sends and receives a packet.
- void set_txpacket_id (int id)

  Sets the sending packet ID.
- void set_txpacket_instruction (int instruction)

  Sets the sending packet instruction.
- void set_txpacket_parameter (int index, int value)

  Sets the sending packet parameter.
- void set_txpacket_length (int length)

  Sets the sending packet length.
- bool get_rxpacket_error (int error)

  Returns false if no receive error and true if there's an error.
- int get_rxpacket_error_byte (void)

  Returns the error byte.
- int get_rxpacket_parameter (int index)

  Returns the received parameter.
- int get_rxpacket_length ()

  Returns the received packet length.
- void ping (int id)

  Ping to the selected id, check com status for the ping result.
- int read_byte (int id, int address)

  Reads a byte from the selected ID at the selected address.
- void write_byte (int id, int address, int value)

  Writes a byte to the selected ID at the selected address.
- int read_word (int id, int address)

  Reads a word to the selected ID at the selected address.
- void write_word (int id, int address, int value)

  Writes a word to the selected ID at the selected address.
- double get_packet_time ()

Returns the packet time.

- void set_packet_timeout (int NumRcvByte)

  Sets the timeout in number of received bytes.
- void set_packet_timeout_ms (int msec)

  Sets the timeout in ms.
- bool is_packet_timeout ()

  Returns true if the packet is timeout.

Private Attributes

- dxl_hal dH

  Conains the serial port comunication.
- unsigned char gbInstructionPacket [MAXNUM_TXPACKET] = {0}

  Contains all the instructions.
- unsigned char gbStatusPacket [MAXNUM_RXPACKET] = {0}

  Contains the status.
- unsigned int gbRxPacketLength = 0

  Received packet length.
- unsigned int gbRxGetLength = 0

  Temporal length from the received packet.
- double gdPacketStartTime = 0.0

  Packet start time.
- double gdByteTransTime = 0.0

  Byte transmission time.
- double gdRcvWaitTime = 0.0

  Receive wait time.
- int gbCommStatus = COMM_RXSUCCESS

  Current communication status.
- int giBusUsing = 0

  True if the bus if being used.

3.4.1 Detailed Description

Dynamixel 1.0 protocol class.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 dynamixel::dynamixel ( ) [inline]

Default constructor.

```
00097 {}
```

3.4.2.2 dynamixel::dynamixel (QString *port_num*, int *baud_rate* = 1000000)

Initialization constructor.

```
00011 {
00012     initialize(port_num, baud_rate);
00013 }
```

3.4.2.3 dynamixel::~dynamixel ( ) [inline]

Default destructor.

```
00103 { dH.close(); }
```

3.4.3 Member Function Documentation

3.4.3.1 int dynamixel::change_baudrate (int *baud_rate*)

Changes the current baud rate.

```
00031 {
00032     int result = 0;
00033     float baudrate = (float)baud_rate;
00034
00035     result = dH.change_baudrate(baudrate);
00036     if(result == 1)
00037         gdByteTransTime = 1000.0f / baudrate * 10.0; // 1000/baudrate(bit per msec) *
       10(start bit + data bit + stop bit)
00038
00039     return result;
00040 }
```

3.4.3.2 int dynamixel::get_comm_result ( ) [inline]

Returns the current com status.

```
00118 { return gbCommStatus; }
```

3.4.3.3 double dynamixel::get_packet_time (void)

Returns the packet time.

```
00050 {
00051     double elapsed_time;
00052
00053     elapsed_time = (double)(dH.get_curr_time() -
      gdPacketStartTime);
00054
00055     // Overflow
00056     if(elapsed_time < 0) gdPacketStartTime = dH.get_curr_time();
00057
00058     return elapsed_time;
00059 }
```

3.4.3.4 bool dynamixel::get_rxpacket_error (int *error*)

Returns false if no receive error and true if there's an error.

Parameters

error	Selects the error to check

```
00271 {
00272     if( gbStatusPacket[PRT1_PKT_ERRBIT] & (unsigned char)error )
00273         return true;
00274
00275     return false;
00276 }
```

3.4.3.5 int dynamixel::get_rxpacket_error_byte (void)

Returns the error byte.

```
00279 {
00280     return gbStatusPacket[PRT1_PKT_ERRBIT];
00281 }
```

3.4.3.6 int dynamixel::get_rxpacket_length ( )

Returns the received packet length.

```
00289 {
00290     return (int)gbStatusPacket[PRT1_PKT_LENGTH];
00291 }
```

3.4.3.7 int dynamixel::get_rxpacket_parameter (int *index*)

Returns the received parameter.

```
00284 {
00285     return (int)gbStatusPacket[PRT1_PKT_PARAMETER0+index];
00286 }
```

3.4.3.8 int dynamixel::initialize (QString *port_num*, int *baud_rate*)

Initializates the port.

```
00016 {
00017     if( baud_rate < 1900 ) return 0;
00018
00019     if( not dH.open(port_num, baud_rate) ) return false;
00020
00021     // 1000/baudrate(bit per msec) * 10(start bit + data bit + stop bit)
00022     gdByteTransTime = 1000.0 / (double)baud_rate * 10.0;
00023
00024     gbCommStatus = COMM_RXSUCCESS;
00025     giBusUsing = 0;
00026
00027     return true;
00028 }
```

3.4.3.9 bool dynamixel::is_packet_timeout (void)

Returns true if the packet is timeout.

Returns

True if the packet is timeout

```
00074 {
00075     if(this->get_packet_time() > gdRcvWaitTime)
00076         return true;
00077     return false;
00078 }
```

3.4.3.10 bool dynamixel::isOpen ( ) [inline]

True if the port is open.

```
00106 { return dH.isOpen(); }
```

3.4.3.11 void dynamixel::ping (int *id*)

Ping to the selected id, check com status for the ping result.

Parameters

id	ID where the ping is done

```
00294 {
00295     while(giBusUsing);
00296
00297     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00298     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_PING;
00299     gbInstructionPacket[PRT1_PKT_LENGTH] = 2;
00300
00301     txrx_packet();
00302 }
```

3.4.3.12 int dynamixel::read_byte (int *id*, int *address*)

Reads a byte from the selected ID at the selected address.

Parameters

id	Selects the ID to read the byte
address	Selects the address to read the byte

```
00305 {
00306     while(giBusUsing);
00307
00308     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00309     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_READ;
00310     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
00311     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = 1;
00312     gbInstructionPacket[PRT1_PKT_LENGTH] = 4;
00313
00314     txrx_packet();
00315
00316     return (int)gbStatusPacket[PRT1_PKT_PARAMETER0];
00317 }
```

3.4.3.13 int dynamixel::read_word (int *id*, int *address*)

Reads a word to the selected ID at the selected address.

Parameters

id	Selects the ID to read the word
address	Selects the address to read the word

```
00333 {
00334     while(giBusUsing);
00335
00336     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00337     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_READ;
00338     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
00339     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = 2;
00340     gbInstructionPacket[PRT1_PKT_LENGTH] = 4;
00341
00342     txrx_packet();
00343
00344     return MAKEWORD((int)gbStatusPacket[PRT1_PKT_PARAMETER0+0], (int)
      gbStatusPacket[PRT1_PKT_PARAMETER0+1]);
00345 }
```

3.4.3.14 void dynamixel::rx_packet (void)

Receives a packet.

```
00144 {
00145     unsigned char i = 0, j = 0, nRead = 0;
00146     unsigned char checksum = 0;
00147 
00148     if( giBusUsing == 0 )
00149         return;
00150 
00151     if( gbInstructionPacket[PRT1_PKT_ID] == BROADCAST_ID )
00152     {
00153         gbCommStatus = COMM_RXSUCCESS;
00154         giBusUsing = 0;
00155         return;
00156     }
00157 
00158     if( gbCommStatus == COMM_TXSUCCESS )
00159     {
00160         gbRxGetLength = 0;
00161         //gbRxPacketLength = 6; //minimum wait length
00162     }
00163 
00164     while(1)
00165     {
00166         nRead = dH.read( &gbStatusPacket[gbRxGetLength],
    gbRxPacketLength - gbRxGetLength );
00167         gbRxGetLength += nRead;
00168 
00169         if(gbRxGetLength > 4)
00170             gbRxPacketLength = gbStatusPacket[PRT1_PKT_LENGTH] + 4;
00171 
00172         if( gbRxGetLength < gbRxPacketLength )
00173         {
00174             if( is_packet_timeout() == 1 )
00175             {
00176                 if(gbRxGetLength == 0)
00177                     gbCommStatus = COMM_RXTIMEOUT;
00178                 else
00179                     gbCommStatus = COMM_RXCORRUPT;
00180                 giBusUsing = 0;
00181                 return;
00182             }
00183             gbCommStatus = COMM_RXWAITING;
00184             //return;
00185         }
00186         else
00187         {
00188             break;
00189         }
00190     }
00191 
00192     // Find packet header
00193     for( i=0; i<(gbRxGetLength-1); i++ )
00194     {
00195         if( gbStatusPacket[i] == 0xff && gbStatusPacket[i+1] == 0xff )
00196             break;
00197         else if( i == gbRxGetLength-2 && gbStatusPacket[gbRxGetLength-1] == 0xff )
00198             break;
00199         else {
00200             gbCommStatus = COMM_RXCORRUPT;
00201             return;
00202         }
00203     }
00204 
00205     if( i > 0 )
00206     {
00207         for( j=0; j<(gbRxGetLength-i); j++ )
00208             gbStatusPacket[j] = gbStatusPacket[j + i];
00209 
00210         gbRxGetLength -= i;
00211     }
00212 
00213     // Check id pairing
00214     if( gbInstructionPacket[PRT1_PKT_ID] != gbStatusPacket[PRT1_PKT_ID])
00215     {
00216         gbCommStatus = COMM_RXCORRUPT;
00217         giBusUsing = 0;
00218         return;
00219     }
00220 
00221     // Check checksum
00222     for( i=0; i<(gbStatusPacket[PRT1_PKT_LENGTH]+1); i++ )
00223         checksum += gbStatusPacket[i+2];
00224     checksum = ~checksum;
00225 
00226     if( gbStatusPacket[gbStatusPacket[PRT1_PKT_LENGTH]+3] != checksum )
00227     {
00228         gbCommStatus = COMM_RXCORRUPT;
00229         giBusUsing = 0;
```

```
00230         return;
00231     }
00232
00233     gbCommStatus = COMM_RXSUCCESS;
00234     giBusUsing = 0;
00235 }
```

3.4.3.15 void dynamixel::set_packet_timeout (int *NumRcvByte*)

Sets the timeout in number of received bytes.

Parameters

NumRcvByte	Number of received bytes to do a timeout

```
00062 {
00063     gdPacketStartTime = dH.get_curr_time();
00064     gdRcvWaitTime = (gdByteTransTime*(double)NumRcvByte + 2.0*LATENCY_TIME + 2.
      0);
00065 }
```

3.4.3.16 void dynamixel::set_packet_timeout_ms (int *msec*)

Sets the timeout in ms.

Parameters

msec	Miliseconds for the timeout

```
00068 {
00069     gdPacketStartTime = dH.get_curr_time();
00070     gdRcvWaitTime = (double)msec;
00071 }
```

3.4.3.17 void dynamixel::set_txpacket_id (int *id*)

Sets the sending packet ID.

```
00250 {
00251     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00252 }
```

3.4.3.18 void dynamixel::set_txpacket_instruction (int *instruction*)

Sets the sending packet instruction.

```
00255 {
00256     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = (unsigned char)instruction;
00257 }
```

3.4.3.19 void dynamixel::set_txpacket_length (int *length*)

Sets the sending packet length.

```
00266 {
00267     gbInstructionPacket[PRT1_PKT_LENGTH] = (unsigned char)length;
00268 }
```

3.4.3.20 void dynamixel::set_txpacket_parameter (int *index*, int *value*)

Sets the sending packet parameter.

```
00260 {
00261     gbInstructionPacket[PRT1_PKT_PARAMETER0+index] = (unsigned char)value;
00262
00263 }
```

3.4.3.21 int dynamixel::terminate (void)

Closes the comunication.

```
00043 {
00044     dH.close();
00045     return 0;
00046 }
```

3.4.3.22 void dynamixel::tx_packet (void)

Sends a packet.

```
00082 {
00083     unsigned char pkt_idx = 0;
00084     unsigned char TxNumByte, RealTxNumByte;
00085     unsigned char checksum = 0;
00086
00087     if( giBusUsing == 1 )
00088     {
00089         gbCommStatus = COMM_TXFAIL;
00090         return;
00091     }
00092
00093     giBusUsing = 1;
00094
00095     if( gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_PING
00096         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_READ
00097         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_WRITE
00098         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_REG_WRITE
00099         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_ACTION
00100         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_RESET
00101         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_SYNC_WRITE )
00102     {
00103         gbCommStatus = COMM_TXERROR;
00104         giBusUsing = 0;
00105         return;
00106     }
00107
00108     gbInstructionPacket[0] = 0xff;
00109     gbInstructionPacket[1] = 0xff;
00110     for( pkt_idx = 0; pkt_idx < (gbInstructionPacket[PRT1_PKT_LENGTH]+1); pkt_idx++ )
00111         checksum += gbInstructionPacket[pkt_idx+2];
00112     gbInstructionPacket[gbInstructionPacket[PRT1_PKT_LENGTH]+3] = ~
    checksum;
00113
00114     //if( gbCommStatus == COMM_RXTIMEOUT || gbCommStatus == COMM_RXCORRUPT )
00115     //  dH.clear();
00116
00117     dH.clear();
00118
00119     TxNumByte = gbInstructionPacket[PRT1_PKT_LENGTH] + 4;
00120     RealTxNumByte = dH.write( gbInstructionPacket, TxNumByte );
00121
00122     if( TxNumByte != RealTxNumByte )
00123     {
00124         gbCommStatus = COMM_TXFAIL;
00125         giBusUsing = 0;
00126         return;
00127     }
00128
00129     if( gbInstructionPacket[PRT1_PKT_INSTRUCTION] == INST_READ )
00130     {
00131         gbRxPacketLength =  gbInstructionPacket[PRT1_PKT_PARAMETER0+1] + 6;
00132         set_packet_timeout( gbInstructionPacket[PRT1_PKT_PARAMETER0+1] + 6 );
00133     }
```

```
00134     else
00135     {
00136         gbRxPacketLength = 6;
00137         set_packet_timeout( 6 );
00138     }
00139
00140     gbCommStatus = COMM_TXSUCCESS;
00141 }
```

3.4.3.23 void dynamixel::txrx_packet (void)

Sends and receives a packet.

```
00238 {
00239     tx_packet();
00240
00241     if( gbCommStatus != COMM_TXSUCCESS )
00242         return;
00243
00244
00245     rx_packet();
00246 }
```

3.4.3.24 void dynamixel::write_byte (int *id*, int *address*, int *value*)

Writes a byte to the selected ID at the selected address.

Parameters

id	Selects the ID to write the byte
address	Selects the address to write the byte
value	Value to set at the selected location

```
00320 {
00321     while(giBusUsing);
00322
00323     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00324     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_WRITE;
00325     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
00326     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = (unsigned char)value;
00327     gbInstructionPacket[PRT1_PKT_LENGTH] = 4;
00328
00329     txrx_packet();
00330 }
```

3.4.3.25 void dynamixel::write_word (int *id*, int *address*, int *value*)

Writes a word to the selected ID at the selected address.

Parameters

id	Selects the ID to write the word
address	Selects the address to write the word
value	Value to set at the selected location

```
00348 {
00349     while(giBusUsing);
00350
00351     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00352     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_WRITE;
00353     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
00354     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = (unsigned char)LOBYTE(value);
00355     gbInstructionPacket[PRT1_PKT_PARAMETER0+2] = (unsigned char)HIBYTE(value);
00356     gbInstructionPacket[PRT1_PKT_LENGTH] = 5;
00357
00358     txrx_packet();
00359 }
```

3.4.4 Member Data Documentation

3.4.4.1 dxl_hal dynamixel::dH [private]

Conains the serial port comunication.

3.4.4.2 int dynamixel::gbCommStatus = COMM_RXSUCCESS [private]

Current communication status.

3.4.4.3 unsigned char dynamixel::gbInstructionPacket[MAXNUM_TXPACKET] = {0} [private]

Contains all the instructions.

3.4.4.4 unsigned int dynamixel::gbRxGetLength = 0 [private]

Temporal length from the received packet.

3.4.4.5 unsigned int dynamixel::gbRxPacketLength = 0 [private]

Received packet length.

3.4.4.6 unsigned char dynamixel::gbStatusPacket[MAXNUM_RXPACKET] = {0} [private]

Contains the status.

3.4.4.7 double dynamixel::gdByteTransTime = 0.0 [private]

Byte transmission time.

3.4.4.8 double dynamixel::gdPacketStartTime = 0.0 [private]

Packet start time.

3.4.4.9 double dynamixel::gdRcvWaitTime = 0.0 [private]

Receive wait time.

3.4.4.10 int dynamixel::giBusUsing = 0 [private]

True if the bus if being used.

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

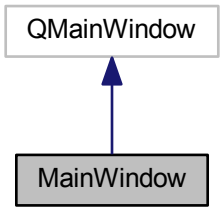
- dxl/dynamixel.h
- dxl/dynamixel.cpp

3.5 MainWindow Class Reference

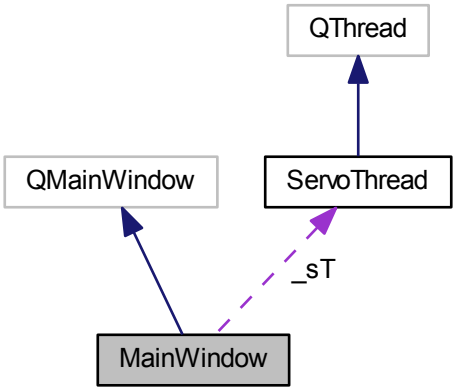
Contains all the windows and other classes.

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Signals

- void joystickChanged ()

  Emmitted when a joystick changes.

Public Member Functions

- MainWindow (QWidget *parent=0)

  Default constructor.
- ~MainWindow ()

  Default destructor.

Private Types

- enum Version { v_1_0 }
- typedef ServoThread::Mode Mode

Private Slots

- void joyChanged ()

  Handles a joystick update.
- void modeChanged (Mode m)

  Handles the change of a mode in the thread.
- void on_actionOptions_triggered ()

  To select the options.
- void on_actionImport_triggered ()

  Opens the import of Dominoes file.
- void on_mode_clicked ()

  Handles the change of the mode.
- void on_reset_clicked ()

  Handles a reset.
- void on_start_clicked ()

  Starts or stops the thread.
- void statusBar (QString s)

  Emits something to the status bar.
- void update ()

  Updates all data to the servo thread.

Private Member Functions

- void keyPressEvent (QKeyEvent *event)

  Handles the press of a key.
- void keyReleaseEvent (QKeyEvent *event)

  Handles the realease of a key.
- void read ()

  Reads the data from the default location.
- void read (QString path)

  Reads the data from the selected path, overloaded function.
- void write ()

  Writes the data to the default location.
- void write (QString path)

  Writes the data to disk overloaded function.

Private Attributes

- QVector< QLabel * > _axis

  Handles all the axis labels.
- QVector< float > _axisV

  Contains the axis value;.
- QVector< QLabel * > _buts

  Handles all the button labels.
- QVector< bool > _butsV

Handles all buttons values.

- QString _dataP

  Contains the path to the data location.

- XJoystick _joy

  To handle the joystick.

- ServoThread _sT

  Contains the thread controlling all the servos and external hardware.

- QTimer _timer

  To update the joystick value.

- Ui::MainWindow * ui

  Contains the user interface.

Static Private Attributes

- static const int sCount = 3

  Contains the number of minimun servos to work.

- static const int aSCount = 0

  Contains the number of additional servos used.

3.5.1 Detailed Description

Contains all the windows and other classes.

3.5.2 Member Typedef Documentation

3.5.2.1 typedef ServoThread::Mode MainWindow::Mode [private]

3.5.3 Member Enumeration Documentation

3.5.3.1 enum MainWindow::Version [private]

Enumerator

v_1_0

```
00034                   {
00035         v_1_0
00036     };
```

3.5.4 Constructor & Destructor Documentation

3.5.4.1 MainWindow::MainWindow (QWidget * *parent* = 0) [explicit]

Default constructor.

```
00005                                       :
00006     QMainWindow(parent),
00007     _axis(XJoystick::AxisCount),
00008     _axisV(XJoystick::AxisCount),
00009     _buts(XJoystick::ButtonCount),
00010     _butsV(XJoystick::ButtonCount),
00011     ui(new Ui::MainWindow)
00012 {
00013     ui->setupUi(this);
00014
00015     connect(&_joy, SIGNAL(changed()), this, SLOT(joyChanged()));
00016     connect(&_timer, SIGNAL(timeout()), this, SLOT(update()));
00017     connect(&_sT, SIGNAL(statusBar(QString)), this, SLOT(statusBar(QString)));
```

```
00018     connect(&_sT, SIGNAL(modeChanged(Mode)), this, SLOT(
      modeChanged(Mode)));
00019
00020
00021     _timer.setInterval(10);
00022     _timer.start();
00023
00024     // JOYSTICK
00025     QVector< QString > V(_joy.getAllAxis());
00026     // Adding axis
00027     QGridLayout *wL = new QGridLayout;
00028     for (int i = 0; i < XJoystick::AxisCount; ++i) {
00029         QHBoxLayout *L = new QHBoxLayout;
00030         L->addWidget(new QLabel(V[i].append(":"), this));
00031         _axis[i] = new QLabel("#");
00032         L->addWidget(_axis[i]);
00033         L->addStretch();
00034         wL->addLayout(L, i%3, i/3);
00035     }
00036     ui->joyAxis->setLayout(wL);
00037
00038     // Adding buttons
00039     wL = new QGridLayout;
00040     for (int i = 0; i < XJoystick::ButtonCount; ++i) {
00041         _buts[i] = new QLabel(QString::number(i + 1));
00042         wL->addWidget(_buts[i], i/8, i%8);
00043         _buts[i]->setEnabled(false);
00044         _buts[i]->hide();
00045     }
00046     ui->joyButs->setLayout(wL);
00047     ui->joyAxis->hide();
00048     ui->joyButs->hide();
00049     ui->line->hide();
00050
00051     // Creating data Path
00052     _dataP = QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
00053     QDir dir(_dataP);
00054     if (!dir.exists()) dir.mkpath(_dataP);
00055
00056     read();
00057     _sT.start();
00058 }
```

3.5.4.2 MainWindow::~MainWindow ( )

Default destructor.

```
00061 {
00062     delete ui;
00063 }
```

3.5.5 Member Function Documentation

3.5.5.1 void MainWindow::joyChanged ( ) [private], [slot]

Handles a joystick update.

```
00110 {
00111     int sel = _joy.current();
00112
00113     QVector< XJoystick::Info > V(_joy.available());
00114     bool found = false;
00115     int i = 0;
00116     while (i < V.size() and not found) { found = V[i].ID == sel; ++i; }
00117     if (not found) {
00118         if (V.size() > 0) {
00119             _joy.select(V[0].ID);
00120             ui->line->hide();
00121
00122             // Showing axis
00123             ui->joyAxis->show();
00124
00125             // Showing buttons
00126             for (QLabel *l : _buts) l->hide();
00127             ui->joyButs->show();
00128             int n = _joy.buttonCount();
00129             for (int i = 0; i < n; ++i) _buts[i]->show();
```

```
00130         }
00131         else {
00132             _joy.select(-1);
00133             ui->joyAxis->hide();
00134             ui->joyButs->hide();
00135             ui->line->hide();
00136         }
00137     }
00138     emit joystickChanged();
00139 }
```

3.5.5.2 void MainWindow::joystickChanged ( ) [signal]

Emmitted when a joystick changes.

3.5.5.3 void MainWindow::keyPressEvent (QKeyEvent * *event*) [private]

Handles the press of a key.

```
00066 {
00067     if (event->isAutoRepeat()) return;
00068     if (event->key() == Qt::Key_A) _joy.axisPress(0, -100);
00069     else if (event->key() == Qt::Key_D) _joy.axisPress(0, 100);
00070     else if (event->key() == Qt::Key_W) _joy.axisPress(1, 100);
00071     else if (event->key() == Qt::Key_S) _joy.axisPress(1, -100);
00072     else if (event->key() == Qt::Key_Q) _joy.axisPress(2, -100);
00073     else if (event->key() == Qt::Key_E) _joy.axisPress(2, 100);
00074     else if (event->key() == Qt::Key_J) _joy.axisPress(3, -100);
00075     else if (event->key() == Qt::Key_K) _joy.axisPress(3, 100);
00076     else if (event->key() == Qt::Key_R) _sT.reset();
00077     else if (event->key() == Qt::Key_Enter) _joy.buttonPress(0, true);
00078
00079     this->update();
00080 }
```

3.5.5.4 void MainWindow::keyReleaseEvent (QKeyEvent * *event*) [private]

Handles the realease of a key.

```
00083 {
00084     if (event->isAutoRepeat()) return;
00085     if (event->key() == Qt::Key_A) _joy.axisRelease(0);
00086     else if (event->key() == Qt::Key_D) _joy.axisRelease(0);
00087     else if (event->key() == Qt::Key_W) _joy.axisRelease(1);
00088     else if (event->key() == Qt::Key_S) _joy.axisRelease(1);
00089     else if (event->key() == Qt::Key_Q) _joy.axisRelease(2);
00090     else if (event->key() == Qt::Key_E) _joy.axisRelease(2);
00091     else if (event->key() == Qt::Key_J) _joy.axisRelease(3);
00092     else if (event->key() == Qt::Key_K) _joy.axisRelease(3);
00093     else if (event->key() == Qt::Key_Enter) _joy.buttonRelease(0);
00094     this->update();
00095 }
```

3.5.5.5 void MainWindow::modeChanged (Mode *m*) [private], [slot]

Handles the change of a mode in the thread.

```
00142 {
00143     qDebug() << int(m);
00144     if (m == Mode::Manual) ui->mode->setText("Manual");
00145     else if (m == Mode::Controlled) ui->mode->setText("Auto");
00146 }
```

3.5.5.6 void MainWindow::on_actionImport_triggered ( ) [private], [slot]

Opens the import of Dominoes file.

```
00165 {
00166     QString caption("Open Dominoes File");
00167     QString dir(QDir::homePath());
00168     QString filter(tr("Dominoes file (*.df)"));
00169 
00170     QString file = QFileDialog::getOpenFileName(this, caption, dir, filter);
00171 
00172     if (!file.size()) return;
00173 
00174     _sT.readPath(file);
00175 }
```

3.5.5.7 void MainWindow::on_actionOptions_triggered ( ) [private], [slot]

To select the options.

```
00150 {
00151     _sT.pause();
00152     ui->start->setText("Start");
00153 
00154     OptionsWindow o(_joy, &_sT, this);
00155 
00156     connect(this, SIGNAL(joystickChanged()), &o, SLOT(
      joystickChanged()));
00157 
00158     if (o.exec()) {
00159         o.storeData();
00160         this->write();
00161     }
00162 }
```

3.5.5.8 void MainWindow::on_mode_clicked ( ) [private], [slot]

Handles the change of the mode.

```
00178 {
00179     if (_sT.isActive()) {
00180         _sT.pause();
00181         ui->start->setText("Start");
00182     }
00183     if (ui->mode->text() == "Manual") {
00184         ui->mode->setText("Auto");
00185         _sT.setMode(Mode::Controlled);
00186     }
00187     else if (ui->mode->text() == "Auto") {
00188         ui->mode->setText("Manual");
00189         _sT.setMode(Mode::Manual);
00190     }
00191 }
```

3.5.5.9 void MainWindow::on_reset_clicked ( ) [private], [slot]

Handles a reset.

```
00194 {
00195     _sT.reset();
00196 }
```

3.5.5.10 void MainWindow::on_start_clicked ( ) [private],[slot]

Starts or stops the thread.

```
00199 {
00200     QString text = ui->start->text();
00201
00202     if (text == "Start") {
00203         _sT.wakeUp();
00204         ui->start->setText("Stop");
00205     }
00206     else if (text == "Stop") {
00207         _sT.pause();
00208         ui->start->setText("Start");
00209     }
00210 }
```

3.5.5.11 void MainWindow::read ( ) [inline],[private]

Reads the data from the default location.

```
00087 { read(_dataP); }
```

3.5.5.12 void MainWindow::read (QString *path*) [private]

Reads the data from the selected path, overloaded function.

```
00098 {
00099     QDir dir(path);
00100     _sT.read(dir.filePath("servo.opts"));
00101 }
```

3.5.5.13 void MainWindow::statusBar (QString *s*) [private],[slot]

Emits something to the status bar.

```
00213 {
00214     ui->statusbar->showMessage(s, 1500);
00215 }
```

3.5.5.14 void MainWindow::update ( ) [private],[slot]

Updates all data to the servo thread.

```
00218 {
00219     // Joystick values
00220     _joy.update();
00221     for (int i = 0; i < XJoystick::AxisCount; ++i) {
00222         float temp = _joy[i];
00223         _axisV[i] = temp;
00224         _axis[i]->setText(QString::number(temp));
00225     }
00226     for (int i = 0; i < XJoystick::ButtonCount; ++i) {
00227         bool temp = _joy.button(i);
00228         _butsV[i] = temp;
00229         _buts[i]->setEnabled(temp);
00230     }
00231
00232     _sT.setData(_axisV, _butsV);
00233
00234     QVector<ServoThread::Servo> servo = _sT.getServosInfo();
00235     QVector4D pos = _sT.getCurrentPos();
00236     QString x = QString::number(pos.x());
00237     QString y = QString::number(pos.y());
```

```
00238     QString z = QString::number(pos.z());
00239     QString rot = QString::number(pos.w());
00240     ui->pos->setText(x + " " + y + " " + z + " " + rot + "°");
00241
00242     // Updating position sliders
00243     ui->servo0S->setValue(servo[0].pos);
00244     ui->servo1S->setValue(servo[1].pos);
00245     ui->servo2S->setValue(servo[2].pos);
00246
00247     // Updating position labels
00248     ui->servo0->setText(QString::number(servo[0].pos));
00249     ui->servo1->setText(QString::number(servo[1].pos));
00250     ui->servo2->setText(QString::number(servo[2].pos));
00251 }
```

3.5.5.15 void MainWindow::write ( ) [inline], [private]

Writes the data to the default location.

```
00093 { write(_dataP); }
```

3.5.5.16 void MainWindow::write (QString *path*) [private]

Writes the data to disk overloaded function.

```
00104 {
00105     QDir dir(path);
00106     _sT.write(dir.filePath("servo.opts"));
00107 }
```

3.5.6 Member Data Documentation

3.5.6.1 QVector< QLabel *> MainWindow::_axis [private]

Handles all the axis labels.

3.5.6.2 QVector< float > MainWindow::_axisV [private]

Contains the axis value;.

3.5.6.3 QVector< QLabel *> MainWindow::_buts [private]

Handles all the button labels.

3.5.6.4 QVector< bool > MainWindow::_butsV [private]

Handles all buttons values.

3.5.6.5 QString MainWindow::_dataP [private]

Contains the path to the data location.

3.5.6.6 XJoystick MainWindow::_joy [private]

To handle the joystick.

3.5.6.7 ServoThread MainWindow::_sT [private]

Contains the thread controlling all the servos and external hardware.

3.5.6.8 QTimer MainWindow::_timer [private]

To update the joystick value.

3.5.6.9 const int MainWindow::aSCount = 0 [static], [private]

Contains the number of additional servos used.

3.5.6.10 const int MainWindow::sCount = 3 [static], [private]

Contains the number of minimun servos to work.

3.5.6.11 Ui::MainWindow* MainWindow::ui [private]

Contains the user interface.

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

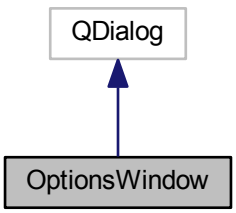
- mainwindow.h
- mainwindow.cpp

3.6 OptionsWindow Class Reference

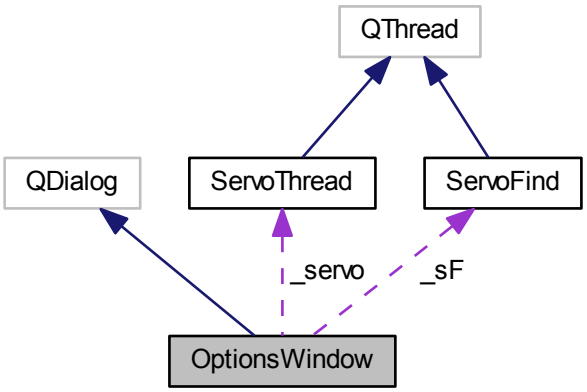
Class used to handle a Window to set the options.

`#include <optionswindow.h>`

Inheritance diagram for OptionsWindow:



Collaboration diagram for OptionsWindow:



Public Slots

- void joystickChanged ()

  To handle the change of a joystick.

Public Member Functions

- OptionsWindow (XJoystick &J, ServoThread ∗servo, QWidget ∗parent=0)

  Default constructor must be intialized with a few values.
- ∼OptionsWindow ()

  Destructor.
- void storeData ()

  Stores all data.

Private Types

- typedef QDialogButtonBox QDB

Private Slots

- void events ()

  Handles events that need to be updated continously.
- void buttonClicked (QAbstractButton ∗but)

  Handles a button clicked.
- void on_servoRefresh_clicked ()

  Refreshes all the servos connected to the port.
- void refreshFinish ()

  Handles the endig of refresh function.

Private Member Functions

- void keyPressEvent (QKeyEvent ∗event)

Private Attributes

- XJoystick & _joy

  Contains the Joystick to handle options.
- int _portSize

  Contains the size of the ports.
- ServoThread ∗ _servo

  Pointer to the servo thread class.
- QVector< QComboBox ∗ > _servoC

  Contains all servo QComboBoxes.
- ServoFind _sF

  Thread to find the servos in a non blocking operation.
- QStatusBar ∗ status

  Status bar.
- QTimer _timer

  Waits for a new COM port.
- Ui::OptionsWindow ∗ ui

  Containsh the GUI.

3.6.1 Detailed Description

Class used to handle a Window to set the options.

3.6.2 Member Typedef Documentation

3.6.2.1 typedef QDialogButtonBox OptionsWindow::QDB [private]

3.6.3 Constructor & Destructor Documentation

3.6.3.1 OptionsWindow::OptionsWindow (XJoystick & *J*, ServoThread ∗ *servo*, QWidget ∗ *parent* = 0) [explicit]

Default constructor must be intialized with a few values.

Parameters

J	Refernce to the Joystick handler
servo	Pointer to the ServoThread
aX	Axis for the X value
aY	Axis for the Y value
aZ	Axis for the Z value

```
00005                                                                    :
00006     QDialog(parent),
00007     _joy(J),
00008     _portSize(-1),
00009     _servo(servo),
00010     _timer(this),
00011     ui(new Ui::OptionsWindow)
00012 {
00013     ui->setupUi(this);
00014
00015     connect(ui->buttonBox, SIGNAL(clicked(QAbstractButton*)),
00016             this, SLOT(buttonClicked(QAbstractButton*)));
```

```
00017
00018     connect(&_sF, SIGNAL(completion(int)),
00019             ui->progressBar, SLOT(setValue(int)));
00020
00021     connect(&_sF, SIGNAL(finished()), this, SLOT(refreshFinish()));
00022
00023     connect(&_timer, SIGNAL(timeout()), this, SLOT(events()));
00024
00025
00026     // Configuring event funcion
00027     _timer.setInterval(500);
00028     _timer.setSingleShot(false);
00029     _timer.start();
00030
00031     status = new QStatusBar(this);
00032     status->setContentsMargins(0, 0, 0, 0);
00033     this->layout()->addWidget(status);
00034
00035     QVector< QString > A(_joy.getAllAxis());
00036
00037     // Updating joystick data
00038     joystickChanged();
00039
00040     // Adding servos
00041     _servoC.push_back(ui->servo0);
00042     _servoC.push_back(ui->servo1);
00043     _servoC.push_back(ui->servo2);
00044     _servoC.push_back(ui->servo3);
00045
00046     for(QComboBox *s : _servoC) s->addItem("None", -1);
00047
00048     QVector<ServoThread::Servo> S(_servo->getServosInfo());
00049     Q_ASSERT(S.size() == _servo->getServosNum());
00050
00051     for (int i = 0; i < S.size(); ++i) {
00052         int ID = S[i].ID;
00053
00054         if (ID >= 0) {
00055             _servoC[i]->addItem(QString::number(ID), ID);
00056             _servoC[i]->setCurrentIndex(1);
00057         }
00058     }
00059
00060     // Obtaining Servo Port information
00061     QString port;
00062     int baud;
00063     _servo->getServoPortInfo(port, baud);
00064     ui->speed->setValue(_servo->getSpeed());
00065     ui->baudRS->setValue(baud);
00066     ui->portS->addItem("", port);
00067 }
```

3.6.3.2 OptionsWindow::~OptionsWindow ( )

Destructor.

```
00070 {
00071     delete ui;
00072     if (_sF.isRunning()) _sF.exit();
00073 }
```

3.6.4 Member Function Documentation

3.6.4.1 void OptionsWindow::buttonClicked (QAbstractButton * *but*) [private],[slot]

Handles a button clicked.

```
00149 {
00150     QDB::ButtonRole role = ui->buttonBox->buttonRole(but);
00151     switch(role) {
00152     case QDB::ApplyRole:
00153         this->storeData();
00154         break;
00155
00156     default:
00157         break;
00158     }
00159 }
```

3.6.4.2 void OptionsWindow::events ( ) [private], [slot]

Handles events that need to be updated continously.

```
00114 {
00115     auto ports = QSerialPortInfo::availablePorts();
00116     ui->portN->setText(QString::number(ports.size()));
00117
00118     if (ports.size() != _portSize) {
00119         _portSize = ports.size();
00120
00121         QString portC(ui->portC->currentData().toString());
00122         QString portS(ui->portS->currentData().toString());
00123
00124         int selC = 0, selS = 0;
00125
00126         ui->portC->clear();
00127         ui->portS->clear();
00128
00129         ui->portC->addItem("None", "");
00130         ui->portS->addItem("None", "");
00131
00132         for (int i = 0; i < ports.size(); ++i) {
00133             QString text(ports[i].portName());
00134             text += ": " + ports[i].description();
00135             ui->portC->addItem(text, ports[i].portName());
00136             ui->portS->addItem(text, ports[i].portName());
00137             if (ports[i].portName() == portC) selC = i + 1;
00138             if (ports[i].portName() == portS) selS = i + 1;
00139         }
00140
00141         if (selS == 0 && ports.size() > 0) selS = 1;
00142
00143         ui->portC->setCurrentIndex(selC);
00144         ui->portS->setCurrentIndex(selS);
00145     }
00146 }
```

3.6.4.3 void OptionsWindow::joystickChanged ( ) [slot]

To handle the change of a joystick.

```
00094 {
00095     // Clear all the items and write the new items
00096     ui->joySel->clear();
00097     ui->joySel->addItem("None", -1);
00098
00099     // Adding items and searching the current
00100     int pos = 0;
00101     QVector<XJoystick::Info> V(_joy.available());
00102     for (int i = 0; i < V.size(); ++i) {
00103         QString text(V[i].name);
00104         text += ": " + QString::number(V[i].ID);
00105         if (V[i].ID == _joy.current()) pos = i;
00106         ui->joySel->addItem(text, V[i].ID);
00107     }
00108     ui->joySel->setCurrentIndex(pos);
00109
00110     ui->joyN->setText(QString::number(V.size()));
00111 }
```

3.6.4.4 void OptionsWindow::keyPressEvent (QKeyEvent * *event*) [private]

```
00179 {
00180     if (event->key() == Qt::Key_Enter || event->key() == Qt::Key_Return) return;
00181     QDialog::keyPressEvent(event);
00182 }
```

3.6.4.5 void OptionsWindow::on_servoRefresh_clicked ( ) [private], [slot]

Refreshes all the servos connected to the port.

```
00162 {
00163     if (_sF.isRunning()) return;
00164     QString port;
00165     int baud;
00166     _servo->getServoPortInfo(port, baud);
00167     int min = ui->min->value();
00168     int max = ui->max->value();
00169     _sF.setData(_servoC, port, baud, min, max);
00170     _sF.start();
00171 }
```

3.6.4.6 void OptionsWindow::refreshFinish ( ) [private],[slot]

Handles the endig of refresh function.

```
00174 {
00175     ui->progressBar->setValue(0);
00176 }
```

3.6.4.7 void OptionsWindow::storeData ( )

Stores all data.

```
00076 {
00077     status->showMessage("Data Stored", 2000);
00078
00079     // Storing joystick data
00080     _joy.select(ui->joySel->currentData().toInt());
00081
00082     QString portS(ui->portS->currentData().toString());
00083     int baudS(ui->baudRS->value());
00084     _servo->setServoPortInfo(portS, baudS);
00085
00086     QVector<int> sID;
00087     for (QComboBox *s : _servoC) sID.push_back(s->currentData().toInt());
00088
00089     _servo->setSID(sID);
00090     _servo->setSpeed(ui->speed->value());
00091 }
```

3.6.5 Member Data Documentation

3.6.5.1 XJoystick& OptionsWindow::_joy [private]

Contains the Joystick to handle options.

3.6.5.2 int OptionsWindow::_portSize [private]

Contains the size of the ports.

3.6.5.3 ServoThread* OptionsWindow::_servo [private]

Pointer to the servo thread class.

3.6.5.4 QVector< QComboBox *> OptionsWindow::_servoC [private]

Contains all servo QComboBoxes.

3.6.5.5 ServoFind OptionsWindow::_sF [private]

Thread to find the servos in a non blocking operation.

3.6.5.6 QTimer OptionsWindow::_timer [private]

Waits for a new COM port.

3.6.5.7 QStatusBar* OptionsWindow::status [private]

Status bar.

3.6.5.8 Ui::OptionsWindow* OptionsWindow::ui [private]

Containsh the GUI.

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

- optionswindow.h
- optionswindow.cpp

3.7 ServoThread::Servo Struct Reference

Struct for the AX12 servos.

`#include <servothread.h>`

Public Member Functions

- Servo (int ID=-1, double pos=-1)
  Default constructor.
- Servo (const Servo &s)
  Copy constructor.
- void operator= (const Servo &s)
  Operator overloading.

Public Attributes

- int ID
  Contains the servo ID.
- double pos
  Contains the servo position.

3.7.1 Detailed Description

Struct for the AX12 servos.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 ServoThread::Servo::Servo (int *ID* = -1, double *pos* = -1) [inline]

Default constructor.

```
00080             : ID(ID), pos(pos) {}
```

3.7.2.2 ServoThread::Servo::Servo (const Servo & *s*) [inline]

Copy constructor.

```
00083 : ID(s.ID), pos(s.pos) {}
```

3.7.3 Member Function Documentation

3.7.3.1 void ServoThread::Servo::operator= (const Servo & *s*) [inline]

Operator overloading.

```
00087         {
00088             this->ID = s.ID;
00089             this->pos = s.pos;
00090         }
```

3.7.4 Member Data Documentation

3.7.4.1 int ServoThread::Servo::ID

Contains the servo ID.

3.7.4.2 double ServoThread::Servo::pos

Contains the servo position.

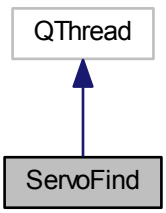
The documentation for this struct was generated from the following file:

- servothread.h

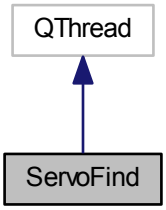
3.8 ServoFind Class Reference

`#include <servofind.h>`

Inheritance diagram for ServoFind:



Collaboration diagram for ServoFind:



Signals

- void completion (int)

  Shows the completion of the process.

Public Member Functions

- ServoFind ()

  Default constructor.
- ~ServoFind ()

  Default destructor.
- void run ()

  Main function.
- void setData (QVector< QComboBox * > servo, QString port, int baud, int min=0, int max=MAX_ID)

  To set all data.

Private Types

- typedef QComboBox QCB

Private Attributes

- int _baud

  Contains the baud rate.
- int _min = 0

  Minimum value to find.
- int _max = MAX_ID

  Maximum value to find.
- QString _port

  Contains the current port.
- QVector< QComboBox * > _servo

  Contains the pointer to the servos QComboBoxes.

3.8.1 Member Typedef Documentation

3.8.1.1 typedef QComboBox ServoFind::QCB [private]

3.8.2 Constructor & Destructor Documentation

3.8.2.1 ServoFind::ServoFind ( )

Default constructor.

```
00004 {
00005
00006 }
```

3.8.2.2 ServoFind::~ServoFind ( )

Default destructor.

```
00009 {
00010
00011 }
```

3.8.3 Member Function Documentation

3.8.3.1 void ServoFind::completion (int) [signal]

Shows the completion of the process.

3.8.3.2 void ServoFind::run ( )

Main function.

```
00014 {
00015     QVector<int> data(_servo.size());
00016
00017     for (int i = 0; i < data.size(); ++i)
00018         data[i] = _servo[i]->currentData().toInt();
00019
00020     for (QCB *s : _servo) {
00021         s->clear();
00022         s->addItem("None", -1);
00023     }
00024
00025     int index = 0;
00026     QVector<int> pos(_servo.size(), 0);
00027
00028     dynamixel dxl(_port, _baud);
00029
00030     for (int i = _min; i < _max; ++i) {
00031         dxl.ping(i);
00032         emit completion(((i - _min)/double(_max - _min))*100.0);
00033         if (dxl.get_comm_result() == COMM_RXSUCCESS) {
00034
00035             for (int j = 0; j < _servo.size(); ++j) {
00036                 if (data[j] == i) pos[j] = index;
00037                 _servo[j]->addItem(QString::number(i), i);
00038             }
00039
00040             ++index;
00041         }
00042     }
00043
00044     for (int i = 0; i < _servo.size(); ++i) _servo[i]->setCurrentIndex(pos[i]);
00045 }
```

3.8.3.3 void ServoFind::setData (QVector< QComboBox * > *servo*, QString *port*, int *baud*, int *min* = 0, int *max* = MAX_ID)

To set all data.

```
00049 {
00050     if (this->isRunning()) return;
00051     _servo = servo;
00052     _port = port;
00053     _baud = baud;
00054
00055     if (min > max) {
00056         int aux = min;
00057         min = max;
00058         max = aux;
00059     }
00060
00061     if (min < 0) min = 0;
00062     if (max > MAX_ID) max = MAX_ID;
00063
00064
00065     _min = min;
00066     _max = max;
00067 }
```

3.8.4 Member Data Documentation

3.8.4.1 int ServoFind::_baud [private]

Contains the baud rate.

3.8.4.2 int ServoFind::_max = MAX_ID [private]

Maximum value to find.

3.8.4.3 int ServoFind::_min = 0 [private]

Minimum value to find.

3.8.4.4 QString ServoFind::_port [private]

Contains the current port.

3.8.4.5 QVector<QComboBox *> ServoFind::_servo [private]

Contains the pointer to the servos QComboBoxes.

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

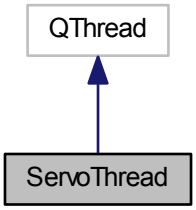
- servofind.h
- servofind.cpp

3.9 ServoThread Class Reference

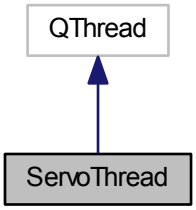
The ServoThread's class handles the comunication between the delta robot servos and the PC.

`#include <servothread.h>`

Inheritance diagram for ServoThread:



Collaboration diagram for ServoThread:



Classes

- struct Dominoe

  Struct to handle the dominoe pieces.
- struct Servo

  Struct for the AX12 servos.

Public Types

- enum Mode { Controlled, Manual, Reset }

  Contains the working mode.

Signals

- void modeChanged (Mode)

  To show the change of a mode.
- void statusBar (QString)

  Emmitted when the status bar must be changed.

Public Member Functions

- ServoThread ()
  Default constructor.
- ~ServoThread ()
  Default destructor.
- void end ()
  Ends the execution.
- QVector4D getCurrentPos ()
  Returns the current position.
- int getServoBaud ()
  Returns the current servo Baud rate.
- QString getServoPort ()
  Returns the current servo Port.
- void getServoPortInfo (QString &port, int &baud)
  Returns both servo Port and baud Rate.
- void getServosInfo (QVector< Servo > &V)
  Returns the servos info, with all its load and current position.
- QVector< Servo > getServosInfo ()
  Overloaded function to get the servo info.
- int getServosNum ()
  Returns the number of servos to handle.
- int getSpeed ()
  Returns the current speed.
- bool isActive ()
  Returns true if the servos are active.
- QMutex * mutex ()
  Returns the mutex used in the thread.
- void pause ()
  Pauses the execution.
- void read (QString file)
  Reads and loads the data from the selected file.
- void readPath (QString file)
  Reads the path where to put the selected pieces.
- void reset ()
  Resets to default positions (used when the mode changes or when some data has changed.
- void setMode (Mode m)
  Sets the current working mode.
- void setData (QVector< float > &aV, QVector< bool > &buts)
  Adds the loaded data.
- void setServoBaud (unsigned int baud)
  Sets the servos port baud rate.
- void setServoPort (QString &port)
  Sets the servos port.
- void setServoPortInfo (QString &port, unsigned int baud)
  Sets the servos port info, data and selected port.
- void setSID (QVector< int > &V)
  Sets the servos ID.
- void setSpeed (unsigned char speed)
  Sets the servos speed.
- void wakeUp ()
  Continues program's execution.
- void write (QString file)
  Writes data to the selected directory.

Private Types

- enum Version { v_1_0 }

  Enum containing all the save file versions.
- enum Status {
  begin, take, waiting, rotate,
  going, ending }

  Contains the available status for the Controlled mode.

Private Member Functions

- bool isPosAvailable (const QVector< Servo > &S, const QVector< double > &D, const QVector3D &newPos, double err)

  Returns true if the position is available.
- bool isReady (const QVector< Servo > &S, const QVector3D &pos, double err)
- void run ()

  Used to create another thread.
- void setAngles (const QVector3D &pos, QVector< double > &D)

  Used to calculate the servos angles.
- double singleAngle (double x0, double y0, double z0)

  Calculates the angle of one servo in the selected position.

Private Attributes

- const double cos60 = 0.5

  Contains the cosinus of 60.
- const double sin60 = sqrt(3)/2

  Contains the sinus of 60.
- const double a = 17.233

  The arm length.
- const double b = 22.648

  The forearm length.
- const double L1 = 6.374

  The base center length.
- const double L2 = 6.000

  The clamp support center lenght.
- const double maxErr = 3.0

  Max available error.
- const double minAngle = 60.0

  Minimum servo angle.
- const double maxAngle = 240.0

  Maximum servo angle.
- const double workRadSq = 144.0

  Working radius squared.
- const uchar ccwCS = 2

  The Counter Clock Wise Compliance Slope.
- const uchar cwCS = 2

  The Clock Wise Compliance Slope.
- QVector4D posStart = QVector4D(11.5, 0.0f, -20, 150)

  Starting position for the controlled mode.
- QVector4D posIdle = QVector4D(0.0f, 0.0f, -20, 150)

Idle position.

- double workHeigh = -25.0

  Working heigh.
- QVector4D _axis

  Contains the axis value.
- QVector< bool > _buts

  Contains the buttons value.
- int _cBaud

  Contains the baud rate used to comunicate with the clamp.
- QWaitCondition _cond

  To start and pause the thread.
- QString _cPort

  Contains the selected com port used to comunitate with the clamp.
- bool _dChanged

  True if the data changes.
- QVector< QVector< Dominoe > > _dominoe

  Contains all the dominoes information.
- bool _end

  True when we must end executino.
- bool _enter

  True if the enter key is pressed.
- Mode _mod

  Contains the working mode.
- QMutex _mutex

  To prevent memory errors between threads.
- bool _pause

  Pauses the execution of the thread.
- QVector4D _pos

  Contains the current position to show to the window.
- int _sBaud

  Contains the used baud rate to comunicate with the servos.
- QVector< Servo > _servos

  Contains the servos information.
- QString _sPort

  Contains the selected com port used in the comunication with servos.
- bool _sPortChanged

  True if the servos port changes.
- unsigned int _sSpeed

  Speed of the robot.
- Status _status

  Current status.

Static Private Attributes

- static const int _sNum = 4

  Number of servos to manage.

3.9.1 Detailed Description

The ServoThread's class handles the comunication between the delta robot servos and the PC.

3.9.2 Member Enumeration Documentation

3.9.2.1 enum ServoThread::Mode

Contains the working mode.

Enumerator

Controlled

Manual

Reset

```
00095     {
00096         Controlled,
00097         Manual,
00098         Reset
00099     };
```

3.9.2.2 enum ServoThread::Status [private]

Contains the available status for the Controlled mode.

Enumerator

begin

take

waiting

rotate

going

ending

```
00028                 {
00029         begin,
00030         take,
00031         waiting,
00032         rotate,
00033         going,
00034         ending
00035     };
```

3.9.2.3 enum ServoThread::Version [private]

Enum containing all the save file versions.

Enumerator

v_1_0

```
00023     {
00024         v_1_0
00025     };
```

3.9.3 Constructor & Destructor Documentation

3.9.3.1 ServoThread::ServoThread ( )

Default constructor.

```
00004                           :
00005     _axis(0, 0, 0, 0),
00006     _buts(XJoystick::ButtonCount),
00007     _cBaud(9600),
00008     _cPort("COM3"),
00009     _dChanged(true),
00010     _end(false),
00011     _mod(Mode::Manual),
00012     _pause(true),
00013     _sBaud(1000000),
00014     _servos(_sNum),
00015     _sPort("COM9"),
00016     _sPortChanged(false),
00017     _sSpeed(30),
00018     _status(Status::begin)
00019 {
00020     for (Servo &s : _servos) s.ID = -1;
00021 }
```

3.9.3.2 ServoThread::~ServoThread ( )

Default destructor.

```
00024 {
00025     _mutex.lock();
00026     _end = true;
00027     _cond.wakeOne();
00028     _mutex.unlock();
00029     wait();
00030 }
```

3.9.4 Member Function Documentation

3.9.4.1 void ServoThread::end ( ) [inline]

Ends the execution.

```
00109     {
00110         _mutex.lock();
00111         _end = true;
00112         _cond.wakeOne();
00113         _mutex.unlock();
00114 
00115         wait();
00116     }
```

3.9.4.2 QVector4D ServoThread::getCurrentPos ( ) [inline]

Returns the current position.

```
00120     {
00121         QMutexLocker m(&_mutex);
00122         return _pos;
00123     }
```

3.9.4.3 int ServoThread::getServoBaud ( ) [inline]

Returns the current servo Baud rate.

```
00127     {
00128         QMutexLocker mL(&_mutex);
00129         return _sBaud;
00130     }
```

3.9.4.4 QString ServoThread::getServoPort ( ) [inline]

Returns the current servo Port.

```
00134     {
00135         QMutexLocker mL(&_mutex);
00136         return _sPort;
00137     }
```

3.9.4.5 void ServoThread::getServoPortInfo (QString & *port*, int & *baud*) [inline]

Returns both servo Port and baud Rate.

```
00141     {
00142         _mutex.lock();
00143         baud = _sBaud;
00144         port = _sPort;
00145         _mutex.unlock();
00146     }
```

3.9.4.6 void ServoThread::getServosInfo (QVector< Servo > & *V*) [inline]

Returns the servos info, with all its load and current position.

Parameters

V	Servo vector to store information

```
00151     {
00152         _mutex.lock();
00153         V = _servos;
00154         _mutex.unlock();
00155     }
```

3.9.4.7 QVector<Servo> ServoThread::getServosInfo ( ) [inline]

Overloaded function to get the servo info.

```
00159     {
00160         QMutexLocker mL(&_mutex);
00161         return _servos;
00162     }
```

3.9.4.8 int ServoThread::getServosNum ( ) [inline]

Returns the number of servos to handle.

```
00165 { return _sNum; }
```

3.9.4.9 int ServoThread::getSpeed ( ) [inline]

Returns the current speed.

```
00169     {
00170         QMutexLocker m(&_mutex);
00171         return _sSpeed;
00172     }
```

3.9.4.10 bool ServoThread::isActive ( ) [inline]

Returns true if the servos are active.

```
00176     {
00177         QMutexLocker m(&_mutex);
00178         return not _pause;
00179     }
```

3.9.4.11 bool ServoThread::isPosAvailable (const QVector< Servo > & *S*, const QVector< double > & *D*, const QVector3D & *newPos*, double *err*) [private]

Returns true if the position is available.

```
00132 {
00133     for (int i = 0; i < 3; ++i) {
00134         double aux = abs(S[i].pos - D[i]);
00135         if (aux > err) return false;
00136     }
00137
00138     if (newPos.toVector2D().lengthSquared() > workRadSq) return false;
00139
00140     QVector<double> theta(3);
00141     this->setAngles(newPos, theta);
00142
00143     for (const double &d : theta) {
00144         if (qIsNaN(d)) return false;
00145         else if (d > maxAngle or d < minAngle) return false;
00146     }
00147
00148     return true;
00149 }
```

3.9.4.12 bool ServoThread::isReady (const QVector< Servo > & *S*, const QVector3D & *pos*, double *err*) [private]

```
00153 {
00154     QVector<double> D(3);
00155     this->setAngles(pos, D);
00156
00157     for (int i = 0; i < 3; ++i) {
00158         double aux = abs(S[i].pos - D[i]);
00159         if (aux > err) return false;
00160     }
00161     return true;
00162 }
```

3.9.4.13 void ServoThread::modeChanged (Mode) [signal]

To show the change of a mode.

3.9.4.14 QMutex* ServoThread::mutex ( ) [inline]

Returns the mutex used in the thread.

```
00182 { return &_mutex; }
```

3.9.4.15 void ServoThread::pause ( ) [inline]

Pauses the execution.

```
00186     {
00187         _mutex.lock();
00188         _pause = true;
00189         _mutex.unlock();
00190     }
```

3.9.4.16 void ServoThread::read (QString *file*)

Reads and loads the data from the selected file.

Parameters

file	Path to the selected file

```
00033 {
00034     // Opening file for reading
00035     QFile f(file);
00036     if (!f.open(QIODevice::ReadOnly)) {
00037         emit statusBar("Cannot read stored data");
00038         return;
00039     }
00040     QDataStream df(&f);
00041
00042     QMutexLocker mL(&_mutex);
00043
00044     int version;
00045     df >> version;
00046     if (version != Version::v_1_0) {
00047         emit statusBar("Error opening file");
00048         return;
00049     }
00050
00051     df >> _cBaud >> _cPort >> _sBaud >> _sPort >>
      _sSpeed;
00052     unsigned int en;
00053     df >> en;
00054     _mod = static_cast<Mode>(en);
00055
00056     int size;
00057     df >> size;
00058     _servos.resize(size);
00059     for (Servo &s : _servos) df >> s.ID;
00060     _dChanged = true;
00061
00062 }
```

3.9.4.17 void ServoThread::readPath (QString *file*)

Reads the path where to put the selected pieces.

Parameters

file	Path to the file where to read the pieces

```
00065 {
00066     // Opening file for reading
00067     QFile f(file);
00068     if (!f.open(QIODevice::ReadOnly)) {
00069         emit statusBar("Error opening file");
00070         return;
00071     }
00072
00073     QTextStream pF(&f);
00074
00075     int size;
00076     pF >> size;
00077     QVector<Dominoe> temp(size);
00078     for (Dominoe &d : temp) pF >> d.X >> d.Y >> d.ori;
00079
00080     _mutex.lock();
00081     double sep = 2; // 2cm of separation
00082     QVector2D ori(12, 0);
00083
00084     for (int i = 0; i < temp.size(); ++i) {
00085         QVector2D aux(temp[i].X, temp[i].Y);
00086         aux -= ori;
00087         int steps = aux.length()/sep;
00088
00089         for (int j = 1; j <= steps; ++j){
00090             Dominoe dAux(j*aux/double(steps) + ori, temp[i].ori);
00091             _dominoe[i].push_back(dAux);
00092         }
00093     }
00094     _dChanged = true;
```

```
00095     _mutex.unlock();
00096
00097     f.close();
00098
00099     emit statusBar("File loaded succesfully");
00100 }
```

3.9.4.18 void ServoThread::reset ( ) [inline]

Resets to default positions (used when the mode changes or when some data has changed.

Precondition

The thread is sleeping

```
00204     {
00205         _mutex.lock();
00206         _mod = Mode::Reset;
00207         _mutex.unlock();
00208     }
```

3.9.4.19 void ServoThread::run ( ) [private]

Used to create another thread.

```
00165 {
00166     // First initializations
00167     _mutex.lock();
00168     int sBaud = _sBaud;
00169     QString sPort = _sPort;
00170     _mutex.unlock();
00171
00172     // Serial port interface
00173     dynamixel dxl(sPort, sBaud);
00174
00175     // Contains the servos comunication
00176     QVector<AX12> A(4);
00177
00178     // Contains the servos angles
00179     QVector<double> D(3);
00180     // First initialization
00181     _mutex.lock();
00182     for (int i = 0; i < A.size(); ++i) {
00183         A[i] = AX12(&dxl);
00184         A[i].setID(_servos[i].ID);
00185         A[i].setSpeed(_sSpeed);
00186         A[i].setComplianceSlope(ccwCS, cwCS);
00187     }
00188     _mutex.unlock();
00189
00190     // Contains the current servo data
00191     QVector< Servo > S(_sNum);
00192
00193     QVector4D pos(posIdle);
00194     QVector4D axis(0, 0, 0, 0);
00195     QVector< bool > buts;
00196
00197     // Contains the domino number to put
00198     unsigned int dom = 0;
00199     unsigned int pas = 0;
00200     QVector< QVector< Dominoe > > Dom;
00201
00202     while (not _end) {
00203         _mutex.lock();
00204
00205         // Pause
00206         if (not _end and _pause) {
00207             dxl.terminate();
00208             _cond.wait(&_mutex);
00209             dxl.initialize(sPort, sBaud);
00210         }
00211
00212         // Data changed handle
00213         if (_dChanged) {
00214             if (sPort != _sPort or sBaud != _sBaud) {
```

```
00215                 sPort = _sPort;
00216                 sBaud = _sBaud;
00217                 dxl.terminate();
00218                 dxl.initialize(sPort, sBaud);
00219             }
00220
00221             for (int i = 0; i < S.size(); ++i) {
00222                 A[i].setID(_servos[i].ID);
00223                 A[i].setSpeed(_sSpeed);
00224                 A[i].setComplianceSlope(ccwCS, cwCS);
00225             }
00226
00227             Dom = _dominoe;
00228             dom = 0;
00229
00230             pos = posIdle;
00231             this->setAngles(pos.toVector3D(), D);
00232             for (int i = 0; i < 3; ++i) A[i].setGoalPosition(D[i]);
00233
00234             _dChanged = false;
00235         }
00236
00237         for (int i = 0; i < A.size(); ++i) {
00238             _servos[i].pos = S[i].pos = A[i].getCurrentPos();
00239         }
00240         axis = _axis;
00241         buts = _buts;
00242         _pos = pos;
00243         _mutex.unlock();
00244
00245
00246         // Main function with data updated
00247         if (_mod == Mode::Manual) {
00248             QVector4D posAux = pos + 0.5*axis;
00249
00250             bool ok = this->isPosAvailable(S, D, posAux.toVector3D(),
00251                                            maxErr + 4);
00252             if (ok) pos = posAux;
00253         }
00254         else if (_mod == Mode::Controlled) {
00255             switch(_status) {
00256             case Status::begin:
00257                 pos = posStart;
00258                 if (this->isReady(S, pos.toVector3D(), maxErr))
00259                     _status = Status::take;
00260                 break;
00261
00262             case Status::take:
00263                 pos[2] = workHeigh;
00264                 if (this->isReady(S, pos.toVector3D(), maxErr))
00265                     _status = Status::waiting;
00266                 break;
00267
00268             case Status::waiting:
00269                 if (buts[0]) _status = Status::rotate;
00270                 else break;
00271
00272             case Status::rotate:
00273             {
00274                 double angle = Dom[dom][0].ori;
00275                 angle += 150.0;
00276                 if (angle > 180.0) angle -= 180.0;
00277                 A[3].setGoalPosition(angle);
00278                 double aux = abs(S[3].pos - angle);
00279                 if (aux < maxErr) {
00280                     _status = Status::going;
00281                     pas = 0;
00282                 }
00283             }
00284                 break;
00285
00286             case Status::going:
00287
00288                 break;
00289
00290             case Status::ending:
00291
00292                 break;
00293
00294             default:
00295                 _status = Status::begin;
00296
00297             }
00298         }
00299         else if (_mod == Mode::Reset) {
00300             _mod = Mode::Manual;
00301             pos = QVector3D(0, 0, -20);
```

```
00302             dom = 0;
00303         }
00304
00305         this->setAngles(pos.toVector3D(), D);
00306         for (int i = 0; i < 3; ++i) A[i].setGoalPosition(D[i]);
00307         A[3].setGoalPosition(pos.w());
00308     }
00309     dxl.terminate();
00310     exit(0);
00311 }
```

3.9.4.20 void ServoThread::setAngles (const QVector3D & *pos*, QVector< double > & *D*) [private]

Used to calculate the servos angles.

```
00314 {
00315     double x1 = pos.x() + L2 - L1;
00316     double y1 = pos.z();
00317     double z1 = pos.y();
00318     D[0] = singleAngle(x1,y1,z1);
00319
00320     double x2 = pos.y()*sin60 - pos.x()*cos60 + L2 - L1;
00321     double y2 = pos.z();
00322     double z2 = -pos.y()*cos60 - pos.x()*sin60;
00323     D[1] = singleAngle(x2,y2,z2);
00324
00325     double x3 = -pos.y()*sin60 - pos.x()*cos60 + L2 - L1;
00326     double y3 = pos.z();
00327     double z3 = -pos.y()*cos60 + pos.x()*sin60;
00328     D[2] = singleAngle(x3,y3,z3);
00329
00330     for (double &d : D) d = 240 + d*180/M_PI;
00331 }
```

3.9.4.21 void ServoThread::setData (QVector< float > & *aV*, QVector< bool > & *buts*)

Adds the loaded data.

Parameters

aV	Contains the axis values
buts	Contains the buttons values

```
00103 {
00104     _mutex.lock();
00105     // Copying the joystick values
00106     _axis = QVector4D(aV[0], aV[1], aV[2], aV[3]);
00107     _axis.normalize();
00108     _buts = buts;
00109     _mutex.unlock();
00110 }
```

3.9.4.22 void ServoThread::setMode (Mode *m*) [inline]

Sets the current working mode.

Precondition

The thread must be on pause

Parameters

m	Contains the desired working mode

```
00214     {
00215         QMutexLocker mut(&_mutex);
00216         if (!_pause) return;
00217         _mod = m;
00218         _dChanged = true;
00219     }
```

3.9.4.23 void ServoThread::setServoBaud (unsigned int *baud*) [inline]

Sets the servos port baud rate.

Parameters

baud	Positive number containing the baud rate

```
00229     {
00230         _mutex.lock();
00231         _sBaud = baud;
00232         _dChanged = true;
00233         _mutex.unlock();
00234     }
```

3.9.4.24 void ServoThread::setServoPort (QString & *port*) [inline]

Sets the servos port.

Parameters

port	String containing the port name

```
00239     {
00240         _mutex.lock();
00241         _sPort = port;
00242         _dChanged = true;
00243         _mutex.unlock();
00244     }
```

3.9.4.25 void ServoThread::setServoPortInfo (QString & *port,* unsigned int *baud*) [inline]

Sets the servos port info, data and selected port.

Parameters

port	String containing the selected port
baud	Contains the selected baud rate

```
00250     {
00251         _mutex.lock();
00252         _sPort = port;
00253         _sBaud = baud;
00254         _dChanged = true;
00255         _mutex.unlock();
00256     }
```

3.9.4.26 void ServoThread::setSID (QVector< int > & *V*) [inline]

Sets the servos ID.

Parameters

V	Vector containing all the servos ID

```
00262     {
00263         // Error passing the data
00264         if (V.size() != _sNum) {
00265             qDebug() << "Error setting servos";
00266             return;
00267         }
00268 
00269 
00270         _mutex.lock();
00271         for (int i = 0; i < V.size(); ++i) _servos[i].ID = V[i];
00272         _dChanged = true;
00273         _mutex.unlock();
00274     }
```

3.9.4.27 void ServoThread::setSpeed (unsigned char *speed*) [inline]

Sets the servos speed.

Parameters

speed	unsigned char from 0 to 100 containing the % of speed

```
00279     {
00280         if (speed > 100) speed = 100;
00281 
00282         _mutex.lock();
00283         _sSpeed = speed;
00284         _dChanged = true;
00285         _mutex.unlock();
00286     }
```

3.9.4.28 double ServoThread::singleAngle (double *x0*, double *y0*, double *z0*) [private]

Calculates the angle of one servo in the selected position.

```
00334 {
00335     double n = b*b - a*a - z0*z0 - x0*x0 - y0*y0;
00336     double raiz = sqrt (n*n*y0*y0 - 4*(x0*x0 + y0*y0)*(-x0*x0*a*a + n*n/4));
00337 
00338     if (x0 < 0) raiz *= -1;
00339     double y = (-n*y0 + raiz ) / (2*(x0*x0 + y0*y0));
00340 
00341     int signe = 1;
00342     if ((b*b - (y0 + a)*(y0 + a)) < (x0*x0 + z0*z0) && x0 < 0) signe *= -1;
00343     double x = sqrt(a*a - y*y)*signe;
00344     return atan2 (y,x);
00345 }
```

3.9.4.29 void ServoThread::statusBar (QString) [signal]

Emmitted when the status bar must be changed.

3.9.4.30 void ServoThread::wakeUp ( ) [inline]

Continues program's execution.

```
00290     {
00291         _mutex.lock();
00292         _pause = false;
00293         _cond.wakeOne();
00294         _mutex.unlock();
00295     }
```

3.9.4.31 void ServoThread::write (QString *file*)

Writes data to the selected directory.

Parameters

file	Path to the file

```
00113 {
00114     // Opening file for writing
00115     QFile f(file);
00116     f.open(QIODevice::WriteOnly);
00117     QDataStream df(&f);
00118
00119     _mutex.lock();
00120
00121     // Clamp and servos baud rate and port must be writen
00122     df << int(Version::v_1_0) << _cBaud << _cPort << _sBaud <<
      _sPort << _sSpeed
00123        << int(_mod) << _servos.size();
00124     for (const Servo &s : _servos) df << s.ID;
00125
00126     _mutex.unlock();
00127 }
```

3.9.5 Member Data Documentation

3.9.5.1 QVector4D ServoThread::_axis [private]

Contains the axis value.

3.9.5.2 QVector< bool > ServoThread::_buts [private]

Contains the buttons value.

3.9.5.3 int ServoThread::_cBaud [private]

Contains the baud rate used to comunicate with the clamp.

3.9.5.4 QWaitCondition ServoThread::_cond [private]

To start and pause the thread.

3.9.5.5 QString ServoThread::_cPort [private]

Contains the selected com port used to comunitate with the clamp.

3.9.5.6 bool ServoThread::_dChanged [private]

True if the data changes.

3.9.5.7 QVector< QVector< Dominoe > > ServoThread::_dominoe [private]

Contains all the dominoes information.

3.9.5.8 bool ServoThread::_end [private]

True when we must end executino.

3.9.5.9 bool ServoThread::_enter [private]

True if the enter key is pressed.

3.9.5.10 Mode ServoThread::_mod [private]

Contains the working mode.

3.9.5.11 QMutex ServoThread::_mutex [private]

To prevent memory errors between threads.

3.9.5.12 bool ServoThread::_pause [private]

Pauses the execution of the thread.

3.9.5.13 QVector4D ServoThread::_pos [private]

Contains the current position to show to the window.

3.9.5.14 int ServoThread::_sBaud [private]

Contains the used baud rate to comunicate with the servos.

3.9.5.15 QVector< Servo > ServoThread::_servos [private]

Contains the servos information.

3.9.5.16 const int ServoThread::_sNum = 4 [static], [private]

Number of servos to manage.

3.9.5.17 QString ServoThread::_sPort [private]

Contains the selected com port used in the comunication with servos.

3.9.5.18 bool ServoThread::_sPortChanged [private]

True if the servos port changes.

3.9.5.19 unsigned int ServoThread::_sSpeed [private]

Speed of the robot.

3.9.5.20 Status ServoThread::_status [private]

Current status.

3.9.5.21 const double ServoThread::a = 17.233 [private]

The arm length.

3.9.5.22 const double ServoThread::b = 22.648 [private]

The forearm length.

3.9.5.23 const uchar ServoThread::ccwCS = 2 [private]

The Counter Clock Wise Compliance Slope.

3.9.5.24 const double ServoThread::cos60 = 0.5 [private]

Contains the cosinus of 60.

3.9.5.25 const uchar ServoThread::cwCS = 2 [private]

The Clock Wise Compliance Slope.

3.9.5.26 const double ServoThread::L1 = 6.374 [private]

The base center length.

3.9.5.27 const double ServoThread::L2 = 6.000 [private]

The clamp support center lenght.

3.9.5.28 const double ServoThread::maxAngle = 240.0 [private]

Maximum servo angle.

3.9.5.29 const double ServoThread::maxErr = 3.0 [private]

Max available error.

3.9.5.30 const double ServoThread::minAngle = 60.0 [private]

Minimum servo angle.

3.9.5.31 QVector4D ServoThread::posIdle = QVector4D(0.0f, 0.0f, -20, 150) [private]

Idle position.

3.9.5.32 QVector4D ServoThread::posStart = QVector4D(11.5, 0.0f, -20, 150) [private]

Starting position for the controlled mode.

3.9.5.33 const double ServoThread::sin60 = sqrt(3)/2 [private]

Contains the sinus of 60.

3.9.5.34 double ServoThread::workHeigh = -25.0 [private]

Working heigh.

3.9.5.35 const double ServoThread::workRadSq = 144.0 [private]

Working radius squared.

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

- servothread.h
- servothread.cpp

Chapter 4

File Documentation

4.1 dxl/ax12.cpp File Reference

Contains the AX12 class implementation.

4.1.1 Detailed Description

Contains the AX12 class implementation.

4.2 dxl/ax12.h File Reference

Contains the AX12 class declaration.

Classes

- class AX12

  The AX12 class is used to control AX-12 motors from Dynamixel.

4.2.1 Detailed Description

Contains the AX12 class declaration.

4.3 dxl/dxl_hal.cpp File Reference

Contains the Dynamixel SDK platform dependent header source.

4.3.1 Detailed Description

Contains the Dynamixel SDK platform dependent header source.

4.4 dxl/dxl_hal.h File Reference

Contains the Dynamixel SDK platform dependent header declaration.

Classes

- class dxl_hal

  Dynamixel SDK platform dependent.

4.4.1 Detailed Description

Contains the Dynamixel SDK platform dependent header declaration.

4.5 dxl/dynamixel.cpp File Reference

Contains the dynamixel class implementation.

4.5.1 Detailed Description

Contains the dynamixel class implementation.

4.6 dxl/dynamixel.h File Reference

Contains the dynamixel class declaration.

Classes

- class dynamixel

  Dynamixel 1.0 protocol class.

4.6.1 Detailed Description

Contains the dynamixel class declaration.

4.7 main.cpp File Reference

Contains the Main of the program.

Functions

- int main (int argc, char ∗argv[ ])

4.7.1 Detailed Description

Contains the Main of the program.

4.7.2 Function Documentation

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

```
00009 {
00010     QApplication a(argc, argv);
00011     MainWindow w;
00012     w.show();
00013     return a.exec();
00014 }
```

4.8 mainwindow.cpp File Reference

Contains the MainWindow class implementation.

4.8.1 Detailed Description

Contains the MainWindow class implementation.

4.9 mainwindow.h File Reference

Contains the MainWindow class declaration.

Classes

- class MainWindow

  Contains all the windows and other classes.

Namespaces

- Ui

  Namespace to work with a User Interface Qt Form.

4.9.1 Detailed Description

Contains the MainWindow class declaration.

4.10 optionswindow.cpp File Reference

Contains the OptionsWindow class implementation.

4.10.1 Detailed Description

Contains the OptionsWindow class implementation.

4.11 optionswindow.h File Reference

Contains the OptionsWindow class declaration.

Classes

- class OptionsWindow

  Class used to handle a Window to set the options.

Namespaces

- Ui

  Namespace to work with a User Interface Qt Form.

4.11.1 Detailed Description

Contains the OptionsWindow class declaration.

4.12 servofind.cpp File Reference

4.13 servofind.h File Reference

Classes

- class ServoFind

4.14 servothread.cpp File Reference

Contains the ServoThread class implementation.

4.14.1 Detailed Description

Contains the ServoThread class implementation.

4.15 servothread.h File Reference

Contains the ServoThread class declaration.

Classes

- class ServoThread

  The ServoThread's class handles the comunication between the delta robot servos and the PC.
- struct ServoThread::Dominoe

  Struct to handle the dominoe pieces.
- struct ServoThread::Servo

  Struct for the AX12 servos.

4.15.1 Detailed Description

Contains the ServoThread class declaration.

4.16 stable.h File Reference

Contains all includes in a precompiled header.

4.16.1 Detailed Description

Contains all includes in a precompiled header.

The includes are:

- QAbstractButton
- QApplication
- QComboBox
- QElapsedTimer
- QDebug
- QDialog
- QDialogButtonBox
- QDir
- QFileDialog
- QKeyEvent
- QLabel
- QMainWindow
- QMutex
- QSerialPortInfo
- QStandardPaths
- QStatusBar
- QString
- QtGlobal
- QThread
- QTime
- QTimer
- QVector
- QVector3D
- QVector4D
- QWaitCondition
- XJoystick

Index