DYNAMIXEL LIBRARY

MARKUS SCHNEIDER

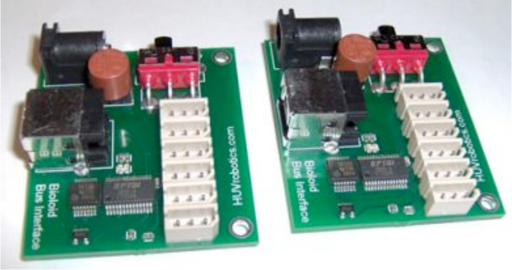
April 2008

Markus Schneider Markus.Schneider@hs-weingarten.de

The Hardware

Bioloid Bus Interface

I order the usb to dynamixel bus interface from http://www.huvrobotics.com. With this board you can directly connect from your PC to dynamixel components. This board uses an FTDI transceiver chip.



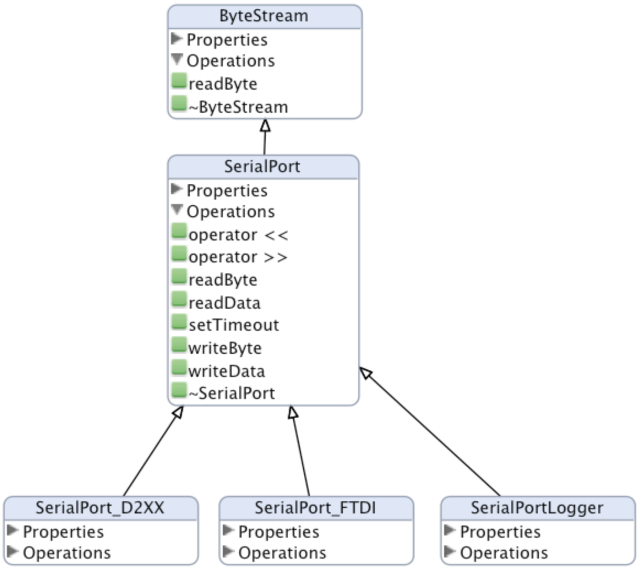
Drivers

I tested 2 different drivers. The first one was the **D2XX** from http://www.ftdichip.com/Drivers/D2XX.htm. See the Installation Guides on the homepage for instructions. Unfortunately there are only binary versions of the driver available and therefor it cannot be used for embedded systems like a linux on gumstix for example. There is a second driver - the **libftdi** from http://www.intra2net.com/de/produkte/opensource/ftdi/. The header and sources are already included in my software package. You only need to install **libusb** from http://libusb.wiki.sourceforge.net.

Software Framework

Serial Port

In order to use any arbitrary serial interface there is an abstract class called *SerialPort*. You must simply implement this interface as you need it. Actually there are 3 implementations. *SerialPort_D2XX*, *SerialPort_FTDI* for the D2XX and FTDI driver already discussed and *SerialPortLogger*, a class which could be used to protocol all traffic.



Connection & Packets

There is a class *Packet*, representing the instruction and status packets like described in the AX-12/AX-S1 manuals. These Packets could be transmitted and received trough an instance of class *Connection*:

```
int main (int argc, char * const argv[])
{
  // create serial port
  SerialPort *port = new SerialPort_FTDI();

  // init connection
  Connection conn(port);

  // create a new packet:
  // dynamixel id 9
  // ping command
  // no parameters
  Packet *pkt = new Packet(9, INST_PING, NULL, 0);

  // print instruction packet to stdout: [0xff 0xff 0x9 0x2 0x1 0xf3]
  pkt->fprint(stdout);

  // transmit packet
  conn.write(pkt);
  // read packet
  conn.read(pkt);

  // print statuspacket to stdout: [0xff 0xff 0x9 0x2 0x0 0xf4]
  pkt->fprint(stdout);

  delete port;
  delete pkt;

  return 0;
}
```

AX12 & AXS1

All functionality of the AX-12 servos and the AX-S1 sensor modules are encapsulated in there classes. The name of the methods correspond to the entries in the Control Table of the AX-12/AX-S1 manuals. All read/write operations to this table are wrapped in get and set methods. For each call an instruction packet is sent and a status packet is expected. Therefore you should not change the *Status Return Level* (Address 0x10). Here a simple example that displays the current position and then moves to another:

```
#include <stdio.h>
#include "serialport_d2xx.h"
#include "serialport_ftdi.h"
#include "connection.h"
#include "packet.h"
#include "ax12.h"

int main (int argc, char * const argv[])
{
  int status;

  // create serial port
  SerialPort *port = new SerialPort_FTDI();

  // init connection
  Connection conn(port);

  // Get a new instance of AX12 with id 9
  // all packets should be transferd over the connection
  // initialized before.
  AX12 *m9 = new AX12(9, &conn);

  // get present position of ax12
  word position;
  status = m9->getPresentPosition(&position);

  // check for errors
  if( status < 0 ){
    fprintf(stderr, "[ERROR]: %d", status);
    return -1;
  }

  // print present position
  printf("Position of AX12 with id %d is %d", m9->getId(), position);
  // move to position 0x200 with speed 0x50
  status = m9->setGoalPositionSpeed(0x200, 0x50);

  // check for errors
  if( status < 0 ){
    fprintf(stderr, "[ERROR]: %d", status);
    return -1;
  }

  delete port;
  delete m9;

  return 0;
}
```

Logging

There is a class *SerialPortLogger* that simply log all communication to a filestream. It could be "wrapped around" a *SerialPort* instance.

```
...
#include "serialportlogger.h"

int main (int argc, char * const argv[])
{
  int status;

  // create serial port
  SerialPort *port = new SerialPort_FTDI();

  // logs all to stderr
  // SerialPort *logger = new SerialPortLogger(port, stderr);

  // logs all to the file `log.txt` and don't append on fopen()
  SerialPort *logger = new SerialPortLogger(port, "log.txt", false);

  // init connection
  Connection conn(logger);
  ...
```

Sync Write

It is also possible to use Bioloids Sync Write feature. This is used for controlling many Dynamixels at the same time. Many instructions can be transmitted by a single packet. Only instructions with the same length and addresses of the control table could be used. To create such a packet there is a class *SyncWritePacket*. You can send this through the same way like a normal packet described before, but there will be no status packet.

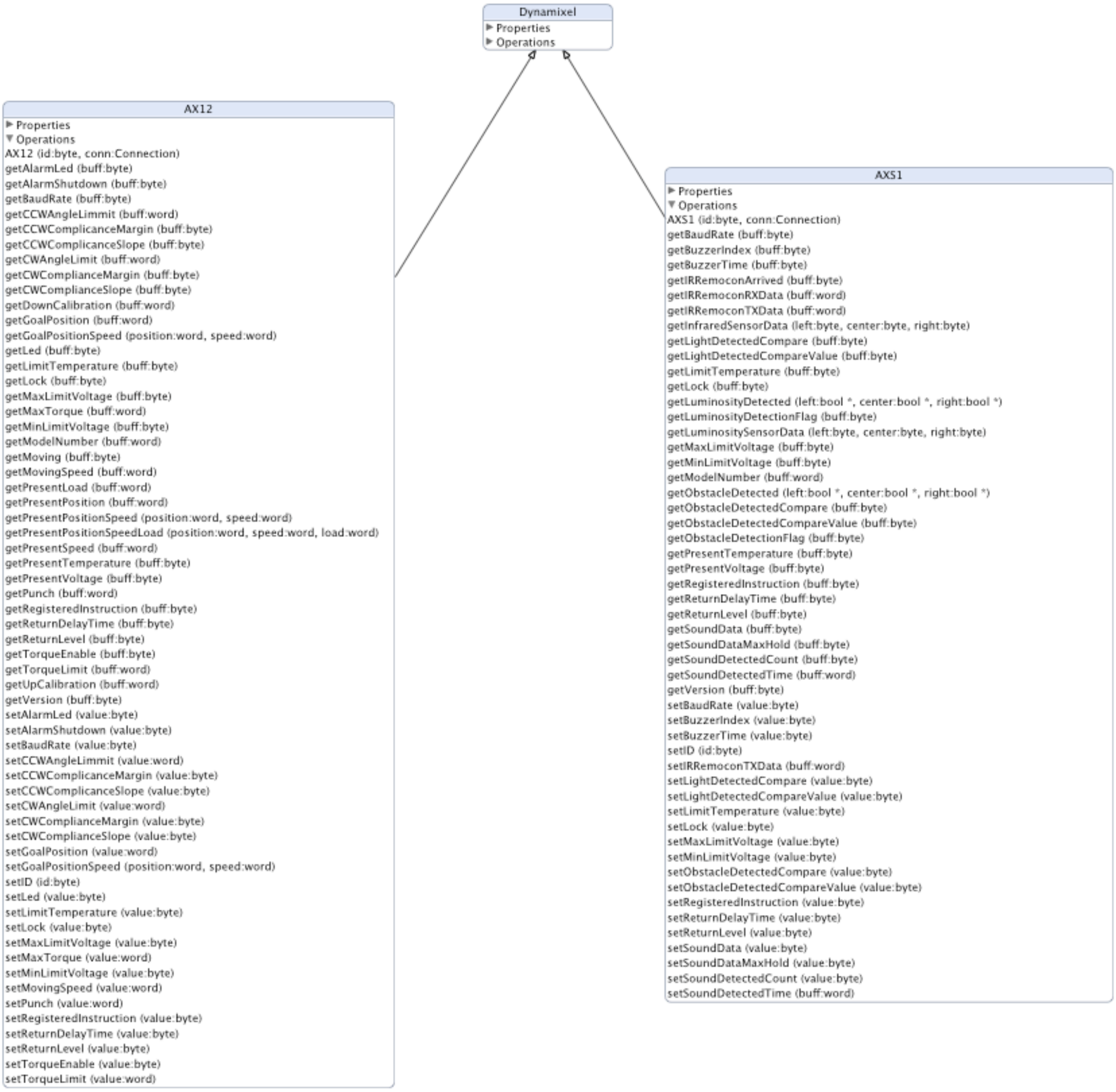
Most of the time goal position and speed needs to send, therefore i created a more high level class, *AX12SyncPositionSpeed*, to do this:

```
// init connection
AX12SyncPositionSpeed swp(&conn);

// new packet:
swp.add(9, 300, 0x100); // set AX12 with id 9 to pos 300 with speed 0x100
swp.add(2, 10, 0x10);   // set AX12 with id 2 to pos  10 with speed 0x10
swp.execute();          // send packet now.

// new packet:
swp.add(9, 100, 0x100); // set AX12 with id 9 to pos 100 with speed 0x100
swp.add(2, 180, 0x30);  // set AX12 with id 2 to pos 180 with speed 0x30
swp.execute();          // send packet now.
```

AX12 / AXS1 Classes


Dynamixel
▶ Properties
▶ Operations
AX12
▶ Properties
▼ Operations
AX12 (id:byte, conn:Connection)
getAlarmLed (buff:byte)
getAlarmShutdown (buff:byte)
getBaudRate (buff:byte)
getCCWAngleLimmit (buff:word)
getCCWComplicanceMargin (buff:byte)
getCCWComplicanceSlope (buff:byte)
getCWAngleLimit (buff:word)
getCWComplianceMargin (buff:byte)
getCWComplianceSlope (buff:byte)
getDownCalibration (buff:word)
getGoalPosition (buff:word)
getGoalPositionSpeed (position:word, speed:word)
getLed (buff:byte)
getLimitTemperature (buff:byte)
getLock (buff:byte)
getMaxLimitVoltage (buff:byte)
getMaxTorque (buff:word)
getMinLimitVoltage (buff:byte)
getModelNumber (buff:word)
getMoving (buff:byte)
getMovingSpeed (buff:word)
getPresentLoad (buff:word)
getPresentPosition (buff:word)
getPresentPositionSpeed (position:word, speed:word)
getPresentPositionSpeedLoad (position:word, speed:word, load:word)
getPresentSpeed (buff:word)
getPresentTemperature (buff:byte)
getPresentVoltage (buff:byte)
getPunch (buff:word)
getRegisteredInstruction (buff:byte)
getReturnDelayTime (buff:byte)
getReturnLevel (buff:byte)
getTorqueEnable (buff:byte)
getTorqueLimit (buff:word)
getUpCalibration (buff:word)
getVersion (buff:byte)
setAlarmLed (value:byte)
setAlarmShutdown (value:byte)
setBaudRate (value:byte)
setCCWAngleLimmit (value:word)
setCCWComplicanceMargin (value:byte)
setCCWComplicanceSlope (value:byte)
setCWAngleLimit (value:word)
setCWComplianceMargin (value:byte)
setCWComplianceSlope (value:byte)
setGoalPosition (value:word)
setGoalPositionSpeed (position:word, speed:word)
setID (id:byte)
setLed (value:byte)
setLimitTemperature (value:byte)
setLock (value:byte)
setMaxLimitVoltage (value:byte)
setMaxTorque (value:word)
setMinLimitVoltage (value:byte)
setMovingSpeed (value:word)
setPunch (value:word)
setRegisteredInstruction (value:byte)
setReturnDelayTime (value:byte)
setReturnLevel (value:byte)
setTorqueEnable (value:byte)
setTorqueLimit (value:word)
AXS1
▶ Properties
▼ Operations
AXS1 (id:byte, conn:Connection)
getBaudRate (buff:byte)
getBuzzerIndex (buff:byte)
getBuzzerTime (buff:byte)
getIRRemoconArrived (buff:byte)
getIRRemoconRXData (buff:word)
getIRRemoconTXData (buff:word)
getInfraredSensorData (left:byte, center:byte, right:byte)
getLightDetectedCompare (buff:byte)
getLightDetectedCompareValue (buff:byte)
getLimitTemperature (buff:byte)
getLock (buff:byte)
getLuminosityDetected (left:bool *, center:bool *, right:bool *)
getLuminosityDetectionFlag (buff:byte)
getLuminositySensorData (left:byte, center:byte, right:byte)
getMaxLimitVoltage (buff:byte)
getMinLimitVoltage (buff:byte)
getModelNumber (buff:word)
getObstacleDetected (left:bool *, center:bool *, right:bool *)
getObstacleDetectedCompare (buff:byte)
getObstacleDetectedCompareValue (buff:byte)
getObstacleDetectionFlag (buff:byte)
getPresentTemperature (buff:byte)
getPresentVoltage (buff:byte)
getRegisteredInstruction (buff:byte)
getReturnDelayTime (buff:byte)
getReturnLevel (buff:byte)
getSoundData (buff:byte)
getSoundDataMaxHold (buff:byte)
getSoundDetectedCount (buff:byte)
getSoundDetectedTime (buff:word)
getVersion (buff:byte)
setBaudRate (value:byte)
setBuzzerIndex (value:byte)
setBuzzerTime (value:byte)
setID (id:byte)
setIRRemoconTXData (buff:word)
setLightDetectedCompare (value:byte)
setLightDetectedCompareValue (value:byte)
setLimitTemperature (value:byte)
setLock (value:byte)
setMaxLimitVoltage (value:byte)
setMinLimitVoltage (value:byte)
setObstacleDetectedCompare (value:byte)
setObstacleDetectedCompareValue (value:byte)
setRegisteredInstruction (value:byte)
setReturnDelayTime (value:byte)
setReturnLevel (value:byte)
setSoundData (value:byte)
setSoundDataMaxHold (value:byte)
setSoundDetectedCount (value:byte)
setSoundDetectedTime (buff:word)