```
/*
+---------------------------------------------------------------------------+
   |          The Mobile Robot Programming Toolkit (MRPT) C++ library
   |
   |
   |                   http://mrpt.sourceforge.net/
   |
   |
   |   Copyright (C) 2005-2010  University of Malaga
   |
   |
   |    This software was written by the Machine Perception and Intelligent
   |      Robotics Lab, University of Malaga (Spain).
   |    Contact: Jose-Luis Blanco  <jlblanco@ctima.uma.es>
   |
   |
   |  This file is part of the MRPT project.
   |
   |
   |     MRPT is free software: you can redistribute it and/or modify
   |     it under the terms of the GNU General Public License as published by
   |     the Free Software Foundation, either version 3 of the License, or
   |     (at your option) any later version.
   |
   |
   |   MRPT is distributed in the hope that it will be useful,
   |     but WITHOUT ANY WARRANTY; without even the implied warranty of
   |     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   |     GNU General Public License for more details.
   |
   |
   |     You should have received a copy of the GNU General Public License
   |     along with MRPT.  If not, see <http://www.gnu.org/licenses/>.
   |
   |
   +---------------------------------------------------------------------------
   + */

#include <mrpt/hwdrivers.h> // Precompiled headers

#include <mrpt/hwdrivers/CLMS100eth.h>
#include <mrpt/system/string_utils.h>
```

```
#include <iostream>
#include <sstream>
#include <string.h>

#define APPERTURE           4.712385    // in radian <=> 270°

using namespace mrpt;
using namespace mrpt::system;
using namespace mrpt::utils;
using namespace mrpt::hwdrivers;
using namespace std;


IMPLEMENTS_GENERIC_SENSOR(CLMS100Eth,mrpt::hwdrivers)

CLMS100Eth::CLMS100Eth(string _ip, unsigned int _port):
        m_ip(_ip),
        m_port(_port),
        m_client(),
        m_turnedOn(false),
        m_cmd(),
        m_connected(false),
        m_sensorPose(0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
        m_maxRange(20.0),
        m_beamApperture(.25*M_PI/180.0)
{
}

CLMS100Eth::~CLMS100Eth()
{
    if(m_connected)
        m_client.close();
//    delete m_client;
//    delete m_sensorPose;
}
void CLMS100Eth::loadConfig_sensorSpecific( const mrpt::utils::CConfigFileBase
&configSource,
                            const std::string    &iniSection )
{
    loadExclusionAreas(configSource, iniSection);
    float pose_x, pose_y, pose_z, pose_yaw, pose_pitch, pose_roll;
    bool faillNotFound = false;
    pose_x = configSource.read_float(iniSection,"pose_x",0,faillNotFound);
    if(faillNotFound) return;
    pose_y = configSource.read_float(iniSection,"pose_y",0,faillNotFound);
    if(faillNotFound) return;
    pose_z = configSource.read_float(iniSection,"pose_z",0,faillNotFound);
    if(faillNotFound) return;
    pose_yaw = configSource.read_float(iniSection,"pose_yax",0,faillNotFound);
    if(faillNotFound) return;
    pose_pitch =
    configSource.read_float(iniSection,"pose_pitch",0,faillNotFound);
    if(faillNotFound) return;
    pose_roll =
    configSource.read_float(iniSection,"pose_roll",0,faillNotFound);
    if(faillNotFound) return;

    m_sensorPose = CPose3D( pose_x, pose_y, pose_z,
```

```
        DEG2RAD( pose_yaw ),DEG2RAD( pose_pitch ), DEG2RAD( pose_roll ));

}

bool CLMS100Eth::checkIsConnected(void)
{
    if(m_connected) return true;
    else
    {
        try{
            m_client.connect(m_ip, m_port);
        }catch(std::exception &e)
        {
            printf_debug(e.what());
            printf_debug("[CLMS100ETH] ERROR TRYING TO OPEN Ethernet DEVICE");
            return false;
        }
    }
    m_connected = true;
    return true;
}

bool CLMS100Eth::turnOff()
{
    if(m_client.isConnected())
        m_client.close();
    m_connected = false;
    m_turnedOn = false;
    return true;
}

bool CLMS100Eth::turnOn()
{
    /** From the LMS100 datasheet : :
     * * Login sMN SetAccessMode 03 F4724744F4724744
     * * Set Scanarea and Resolution
     * * sMN mLMPsetscancfg
     * * SWN LMDscandatacfg 01 00 0 1 0 00 00 0 0 0 0 1
     * * SMN mEEwriteall   Je ne le fais pas, car ca écrit en mémoire non
    volatile...
     * * Request scan : sRN LMDscandata OR sEN LMDscandata
     */
    size_t read;
    if(checkIsConnected())
    {
        try{
            {
                char msg[] = {"sMN SetAccessMode 03 F4724744"};
                char msgIn[100];
                sendCommand(msg);
                read = m_client.readAsync(msgIn, 100, 100, 100);  //18
                printf_debug("message : %s\n",string(msgIn).c_str());
                if(!read) return false;
            }
            {
                char msg[] = {"sMN mLMPsetscancfg 2500 1 2500 -450000
                +2250000"};
                char msgIn[100];
                sendCommand(msg);
```

```
                m_client.readAsync(msgIn, 100, 100, 100);
                printf_debug("message : %s\n",string(msgIn).c_str());
                if(!read) return false;
            }
            {
                char msg[] = {"sWN mLMDscandatacfg 01 00 0 1 0 00 00 0 0 0 0
                +1"};
                char msgIn[100];
                sendCommand(msg);
                m_client.readAsync(msgIn, 100, 100, 100);
                printf_debug("message : %s\n",string(msgIn).c_str());
                if(!read) return false;
            }
            m_turnedOn = true;
        }catch(std::exception &e)
        {
            printf_debug(e.what());
            return false;
        }
    }else
    {
        return false;
    }
    return true;
}

void CLMS100Eth::sendCommand(const char *cmd)
{
    generateCmd(cmd);
    if (!m_cmd.empty()) // one never knows...
        m_client.writeAsync(&m_cmd[0], m_cmd.size());
}

/** Add the start and end character.
 */
void CLMS100Eth::generateCmd(const char *cmd)
{
    if(strlen(cmd) > 995)
    {
        printf_debug("la commande est trop longue\n");
        return;
    }
    m_cmd = format("%c%s%c",0x02,cmd,0x03);
}


bool CLMS100Eth::decodeScan(char* buff, CObservation2DRangeScan&
outObservation)
{
    char *next;
    unsigned int idx = 0;
    unsigned int scanCount=0;
    char* tmp;
    double factor;

    next = strtok(buff, " ", &tmp);


    while(next && scanCount==0)
```

```
    {
        //cout << "Interpreting : " << next << endl;
        switch(++idx)
        {
        case 1:
            if(strncmp(&next[1], "sRA", 3) && strncmp(&next[1], "sSN", 3))
            return false;
            break;
        case 2 :
            if(strcmp(next, "LMDscandata")) return false;
            break;
        case 6 :
            if(strcmp(next, "0"))
            {
                THROW_EXCEPTION("STATUS error on LMS100");
                return false;
            }
            printf_debug("STATUS Ok.\n");
            break;
        case 21 :
            if(strcmp(next, "DIST1"))
            {
                THROW_EXCEPTION("LMS100 is not configured to send ditances.");
                return false;
            }
            printf_debug("Distance : OK\n");
            break;
        case 22 :
            factor = strtod(next, NULL);
            break;
        case 26 :
            scanCount = strtoul(next, NULL, 16);
            printf_debug("Scan Count : %d\n", scanCount);
            break;
        default :
                break;
    }
        next = strtok(NULL, " ", &tmp);
    }
    outObservation.aperture = APPERTURE;
    outObservation.rightToLeft = false;
    outObservation.stdError = 0.012f;
    outObservation.sensorPose = m_sensorPose;
    outObservation.beamAperture = m_beamApperture;
    outObservation.maxRange = m_maxRange;

    for(unsigned int i = 0 ; i < scanCount ; i++, next = strtok(NULL, " ",
    &tmp))
    {
        outObservation.scan.push_back(double(strtoul(next, NULL, 16))/1000.0);
        outObservation.validRange.push_back(outObservation.scan[i] <=
        outObservation.maxRange);
    }
    return true;
}

void CLMS100Eth::doProcessSimple(bool &outThereIsObservation,
CObservation2DRangeScan &outObservation, bool &hardwareError)
{
```

```
    if(!m_turnedOn)
    {
        hardwareError = true;
        outThereIsObservation = false;
        return;
    }
    hardwareError = false;

    char msg[] = {"sRN LMDscandata"};
    sendCommand(msg);
    char buffIn[4000];
    //size_t read = m_client.readAsync(buffIn, 4000, 100, 100);
    //cout << "read :" << read << endl;
    //while(m_client.readAsync(buffIn, 4000, 100, 100)) cout << "Lit dans le
    vent" << endl;
    m_client.readAsync(buffIn, 4000, 100, 100);

    if(decodeScan(buffIn, outObservation))
    {
        // Do filter:
        this->filterByExclusionAreas( outObservation );
        this->filterByExclusionAngles( outObservation );
        outThereIsObservation = true;
        hardwareError = false;
    }else
    {
        hardwareError = true;
        outThereIsObservation = false;
        printf_debug("doProcessSimple failed\n");
    }

}
```