L2lidar class

V0.3.7 release

Mark Stegall

The c++ L2lidar class provides an interface to the Unitree L2 4D LiDAR hardware.
It is an open-source alternative the the proprietary Unitree lidar SDK2 archive library.
It uses the Qt libraries to provide a transportable source across platforms.
This currently only been tested for the UDP Ethernet interface to the Unitree L2.

Only one instance of this class should be created in application. The Ethernet does not allow multiple connections unless multicasting is used. This has not been tested using multicasting.
The external dependencies are needed by this class and likely used in the parent classes:

- unitree_lidar_protocolsL2.h
- unitree_lidar_utilitiesL2.h

These are modified versions of the original open-source files from the Unitree Lidar SDK2:

- unitree_lidar_protocols.h
- unitree_lidar_utilities.h

L2lidar Class files:

- L2lidar.h
- L2lidar.cpp

Class L2lidar

public:

```
L2lidar(QObject* parent);  // constructor
~L2lidar(); // uses default destructor

// Accessors to retrieve the latest L2 packets
const LidarImuData imu();
const LidarPointDataPacket Pcl3Dpacket();
const Lidar2DPointDataPacket Pcl2Dpacket();
const LidarAckData ack();
const LidarVersionData version();
const LidarTimeStampData timestamp()
```

```
    // packet stats from L2 (only updates when L2 socket connected)
    // These are not actually critical, only for reporting stats
    const uint64_t totalIMU();
    const uint64_t total3D();
    const uint64_t total2D();
    const uint64_t totalACK();
    const uint64_t totalPackets();
    const uint64_t lostPackets();
    const uint64_t totalOther();
    void ClearCounts();  // clears the packet totals

    // L2 commands
    bool LidarStartRotation(void);
    bool LidarStopRotation(void);
    bool LidarReset(void);
    bool LidarGetVersion(void);
    bool SetWorkMode(uint32_t mode);
    bool requestLatencyMeasurement();
    bool sendLatencyID(uint32_t SeqeunceID);

    // UDP ethernet communications

    // this is only to set the UDP parameters in the class
    // It DOES NOT change the L2 configuration settings
    void LidarSetCmdConfig(
            QString srcIP, uint32_t srcPort,
            QString dstIP, uint32_t dstPort);

    // This set the stored UDP configuration on the L2
    // a power cycle is required after this for it to take effect
    bool setL2UDPconfig(
            QString hostIP, uint32_t hostPort,
            QString LidarIP, uint32_t LidarPort);

    bool ConnectL2();  // bind to create, bind socket, connect callback for decode
    void DisconnectL2();   // close socket

signals:
```

void imuReceived();
void PCL3DReceived();
void PCL2DReceived();
void versionReceived();
void timestampReceived();
void ackReceived();
void latencyMeasured(double ms);

Signals are callbacks used to notify a subscriber that an event has occurred.

The class public methods

Class constructor/destructor
L2lidar::L2lidar(QObject* parent);
~L2lidar::L2lidar();

Stats methods

Packet stats from L2 (only updates when L2 socket connected.)
These are not actually critical, only for reporting stats.

const uint64_t totalIMU();
This return the total number of IMU packets received since the last ClearCounts().

const uint64_t total3D();
This returns the total number of 3D packets received since the last ClearCounts().

const uint64_t total2D();
This returns the total number of 2D packets received since the last ClearCounts().

const uint64_t totalACK();
This returns the total number of ACK packets received since the last ClearCounts().

const uint64_t totalPackets();
This returns the total number of packets received since the last ClearCounts().

const uint64_t lostPackets();

This returns the total number of lost packets received since the last ClearCounts().

const uint64_t totalOther();
This returns the total number of other packets received since the last ClearCounts().

void L2lidar::ClearCounts();
This resets the packet count statistics.

Retrieve packet methods

These are used to retrieve the latest packet data. These are used by the caller after receiving a signal notification that a new packet was received for that type of packet. These calls are thread safe.

const LidarImuData imu();
Get the last IMU packet received.

const LidarPointDataPacket Pcl3Dpacket();
Get the last 3D point cloud packet received.

const Lidar2DPointDataPacket Pcl2Dpacket();
Get the last 3D point cloud packet received.

const LidarAckData ack();
Get the last ACK packet received.

const LidarVersionData version();
Get the last VERSION packet received.

const LidarTimeStampData timestamp()
Get the latest timestamp received.

L2 command methods

These are commands sent to the L2.
They are used to:
change the state of the L2
configure L2
reset the L2

request information from the L2 (such as Version info)

bool L2lidar::LidarStartRotation(void);

Sends a run command to the L2.

If the L2 was in standby then it should start scanning it can take >20 seconds to come up to speed and start sending point cloud data.

bool L2lidar::LidarStopRotation(void);

Send a standby command to the L2.

This causes the L2 to stop the motors and go into low power mode.

Issues have been seen with not being able to bring the L2 out of standby mode without a power cycle.

bool L2lidar::LidarReset(void)

This causes the L2 to restart (should be equivalent to power cycle).

Some 'workmode' changes are only effective after a restart.

Note:

The L2 restart is immediate and does not send and ACK packet that confirms receipt of the command.

bool L2lidar::LidarGetVersion(void);

This sends a request to the L2 for a version packet.

The are 3 posible results:

1. The command is completely ignored (This can happen early in the startup.)
2. An ACK packet is sent with the status: ACK_WAIT_ERROR. This occurs when the L2 is not fully initializes. It will not be fully initialize until the L2 is no longer in standby and the first point cloud packet is sent. If a standby command is sent after the L2 is full initialized then a version packet will still be sent.
3. A VERSION packet is sent (this class will send a signal to any connected subscribers). An ACK packet will also be sent by the L2 with the status ACK_SUCCESS.

bool L2lidar::SetWorkMode(uint32_t mode);

This commands only sets the workmode in the L2. It does not restart the L2. This must be done separately for certain workmode changes.

Note:

Immediately effective workmode settings that have been observed are:

Std/Wide FOV
IMU disable/enable

Effective after restart or reset
2D/3D mode
serial/UPD mode
start automatically or wait for start command after power on

NOTE: This uses an undocumented command to perform this function
It was discovered using Wireshark to see how the Unitree software sends commands. This is how the Unitree software sets workmode.
The define LIDAR_PARAM_WORK_MODE_TYPE was added to be consistent with the documented commands.

void L2lidar::LidarSetCmdConfig(
QString srcIP, uint32_t srcPort,
QString dstIP, uint32_t dstPort);

This command does not communicate with the L2.
It saves the IP setup required to connect to the L2 through UDP.
This will be extended to include the serial com port if UART communications when is implemented.
This must be set before the L2connect() method is used.

bool L2lidar::setL2UDPconfig(
QString hostIP, uint32_t hostPort,
QString LidarIP, uint32_t LidarPort);

This command sets the UDP configuration used by the L2 to send and receive packets through the Ethernet connection to the L2.
Note:
The L2 does not use DHCP. It is really configured for peer to peer operation.
This means the host Ethernet conenction should also be manually configured.
It does not appear that the L2 uses jumbo packets.
The factory default when it is shipped are:
Lidar IP: 192.168.1.62
Lidar port: 6101 (this is the port L2 listens for packets)
Host IP: 192.168.1.2
Host port: 6201 (this is the port the L2 uses to send packets)

If you change these parameters you will also likely need to change the port settings on your host to match.
These settings take effect on the next power cycle of the L2

bool L2lidar::sendLatencyID(uint32_t SeqID);

sets packet sequence ID
This is used in measuring the latency of packets over the UDP interface
It sets a sequence ID that is returned in the ACK packet. This allows an ACK packet to be matched to this specific send command.
Note: Command packets and User command packets use the same packet structure

bool L2lidar::requestLatencyMeasurement();

This request measuremeasurement of the packet latency from the L2 in msec.
-1.0 is invalid measurment
A signal is emitted using latencyMeasured(double latencyMs)
The requester can use a slot connection to obtain the value once it it measured.
The connect looks like:

```
connect(&l2lidar, &L2lidar::latencyMeasured,
        this, &MainWindow::SaveLatency);
```

SaveLatency should look like:

```
void SaveLatency(double measurement) { ....}
```

L2 connect/disconnect methods

These are the equivalent to opening or closing the L2 communications.
To open communications with the L2 you must:

LidarSetCmdConfig()
ConnectL2()

To close you should:

DisconnectL2()

Note: The interface is automatically closed when the L2lidar class is deleted.

bool L2lidar::ConnectL2();

Currently only UDP Ethernet communications is supported.
The UART implementation is only a placeholder for future implementation.
You should call LidarSetCmdConfig() before calling this.
If you do not the factory default settings are used.

NOTE: If you are using WSL2 on Windows 11 then you are also likely to have configured a virtual ethernet port. Both your actual physical Ethernet port and the Virtual Ethernet ports should be manually configured. You can not set both to be the same IP address.

void L2lidar::DisconnectL2()

This closes the UDP socket. The caller can not receive any data from the L2 until a ConnectL2() is performed again.

Signal Methods

These methods are callback functions that are emitted to tell the subscriber that a new packet for that particular type was received. They are notifications only and have no parameters. The use the connect method in the subscriber. The following is an example:

```
connect(&l2lidar, &::L2lidar::PCL3DReceived,
        this, &MainWindow::onNew3DLidarFrame, Qt::QueuedConnection);
```

This calls the method onNew3DLirdarFrame() when the PCL3DRecieved() notification is emitted from the L2lidar class. In response the onNew3DLirdarFrame() would call the Pcl3Dpacket() and possibly the imu() methods to obtain those latest packets. These calls are thread safe.

void imuReceived();	IMU packet received
void PCL3DReceived();	3D point cloud packet received
void PCL2DReceived();	2D point cloud packet received
void versionReceived();	VERSION packet received
void timestampReceived();	Time stamp update received
void ackReceived();	ACK packet received
void latencyMeasured(double ms);	result from a latency measurement request

The class private methods

```
//=====================================================================
//  start of received packet handling section
//  This handles the receipt of packets from the L2
//  This includes error checking of the packets
//  When certain packets are decoded a signal is emitted
//  to conneted subscribers
//  The packet types or conditions that emit signals are:
//     3D point cloud packets
//     2D point cloud packets
//     ACK packet
//     VERSION packet
//     Time stamp updates from IMU, point cloud or timestamp packets
//=====================================================================

//-------------------------------------------------------
void L2lidar::readUDPpendingDatagrams();
//
//  It is a Qt non-blocking I/O service called when data
//  is received on the ethernet interface.
//  This is a callback function that receives the UDP
//  datagram packets.
//
//  It is connected to the readyread Qt thread in
//  this ConnectL2() in this class.
//
//  It is not used externally from the class
//
//-------------------------------------------------------

//-------------------------------------------------------
void L2lidar::readUARTpendingDatagrams();
//  This is a callback function that receives the UDP
//  data packets.  It is a Qt non-blocking I/O service called
//  when data in received on the UART interface
```

```
//  This is just skeleton fragment, to reserve for
//  future implementation
//-------------------------------------------------------


//-------------------------------------------------------
void L2lidar::processDatagram(const QByteArray& datagram);
//  processing the payload package from the datagram
//  note: more than one packet may be in the datagram
//  Each packet must be processed.
//
//  processing requirements:
//  All but one of the L2 Lidar packets can be contained
//  in one UDP datagram.  Some UDP datagrams contain more
//  than one L2 Lidar packet.
//  Only the L2 Lidar packet for 2D scan data spans multiple
//  UDP datagrams (4 datagrams)
//
//  This means that a processing buffer needs to be appended
//  so that it can span multiple UDP datagrams.
//
//-------------------------------------------------------

//-----------------------------------------------------------------
//  3D point cloud packet Decoder
//-----------------------------------------------------------------
void L2lidar::decode3D(const QByteArray& datagram, uint64_t Offset);

//-----------------------------------------------------------------
//  2D point cloud packet Decoder
//-----------------------------------------------------------------
void L2lidar::decode2D(const QByteArray& datagram, uint64_t Offset);

//-----------------------------------------------------------------
//  IMU packet Decoder
//-----------------------------------------------------------------
void L2lidar::decodeImu(const QByteArray& datagram, uint64_t Offset);
```

```
//-----------------------------------------------------------------
//  VERSION packet Decoder
//-----------------------------------------------------------------
void L2lidar::decodeVersion(const QByteArray& datagram, uint64_t Offset);

//-----------------------------------------------------------------
// TIMESTAMP Decoder
// Currently no timestamp packet has been observed being sent
// by the L2
//-----------------------------------------------------------------
void L2lidar::decodeTimestamp(const QByteArray& datagram, uint64_t Offset);

//-----------------------------------------------------------------
// ACK Decoder
//-----------------------------------------------------------------
void L2lidar::decodeAck(const QByteArray& datagram, uint64_t Offset);

//-----------------------------------------------------------------
// Raw Packet Handler
// This is called when the packet type is NOT:
//        ACK
//        VERSION
//        IMU
//        2D PC
//        3D PC
//-----------------------------------------------------------------
void L2lidar::handleRaw(uint32_t packetType,
                        const QByteArray& datagram, uint64_t Offset);

//-----------------------------------------------------------------
//  setPacketHeader
//  This is just a helper functin to help construct a packet to be
//  sent to the L2.
//-----------------------------------------------------------------
void L2lidar::setPacketHeader(FrameHeader *FrameHeader, uint32_t packet_type, uint32_t
packet_size);

//-----------------------------------------------------------------
```

```
// SendUDPPacket
//---------------------------------------------------------------------
bool L2lidar::SendUDPpacket(uint8_t *Buffer,uint32_t Len);

//---------------------------------------------------------------------
//  SendUARTPacket
//  This is not yet implememted.  It is here just as part of the skeleton
//  required when it is implemented
//---------------------------------------------------------------------
bool L2lidar::SendUARTpacket(uint8_t *Buffer,uint32_t Len);
```