HY-PCBA-02 HID device sensor data frame

HY-PCBA-02 can communicate with windows PC as a standard USB HID device. And IMU data are packaged in a 64 bytes frame.

```
typedef struct
{
    float acc_data[3];
    float gyro_data[3];
    float mag_data[3];
    uint32_t uTick;
    float quaternion_6X[4];
}usb_sensor_data;

typedef struct
{
    usb_message_head message_head;
    usb_sensor_data data;
    uint8_t reserve[4];
}usb_sensor_message;
```

1. "message_head" is to tell if it's an IMU data frame, which size is 4 bytes.
2. "acc_data" is the raw data from accelerometer.
3. "gyro_data" is the raw data from gyroscope.
4. "mag_data" is the raw data from magnetometer.
5. "uTick" is the ticks of each frame, the unit is 1/10 ms, starts from device boot.
6. "quaternion_6X" is the result of IMU fusion(only acc and gyro), in quaternion.

IMU data orders

Host device like windows PC can use raw data of imu to do sensor fusion(6-axis or 9-axis). Or just use quaternion provided by ST fusion algorithm running on STM32.

The directions of accelerometer sensor coordinates are as below,

- acc_data[0] - X
- acc_data[1] - Y
- acc_data[2] - Z

```
               ^  Z
              /
             /
   <--------/
 X          |
            |
            v
              Y
Left-handed coordinate
```

For gyroscope, the data order is

- gyro_data[0] - pitch
- gyro_data[1] - yaw
- gyro_data[2] - roll

For magnetometer, the data order is

- mag_data[0] - north and south strength
- mag_data[1] - up and down strength
- mag_data[2] - east and west strength

For pose, the data order is

- quaternion_6X[0] - x
- quaternion_6X[1] - y
- quaternion_6X[2] - z
- quaternion_6X[3] - w

And the quaternion has a 90-degree rotation about X-axis when HY-PCBA-02 is placed horizontally.

```
Z
^  ^ Y
| /
|/
/--------> X

Right-handed coordinate
```

How to get data frame from HY-PCBA-02 on Windows PC

Here we use a demo project "IMU_Win_Demo" to show how to get the sensor data. The project is tested on Visual Studio 15 2017.

Build

```
cd IMU_Win_Demo
mkdir build
cd build
cmake .. -G "Visual Studio 15 2017 Win64"
```

And then open VS to build and run the demo. The demo is to show how to use IMU data to rotate a cube.

Code Structure

- **/src** main source code of the demo
- **/libs** hid lib, which is used to communicate with HY-PCBA-02 via USB
- **/3rdparty** 3rd party libs, including graphic tools to render in OpenGL and Eigen to dispose the rotation.

Implementation

```
// Open HID device
    CUsbHidDevice *pDevice = CUsbHidDevice::getInstance();
    if (pDevice->OpenDevice() < 0)
    {
        cout << "No device detected, please plug the hid device first." << endl;
    }

// A thread will be created to poll the data from device
DWORD WINAPI CUsbHidDevice::HidRespThreadFunc(LPVOID lpParamter)
{
    unsigned char buf[256];
    while (true)
    {
        if (Polling(buf) > 0)
        {
            RESP_INTERFACE_LIST::iterator iterator;

            for (iterator = m_listHidRespInterface.begin(); iterator != m_listHidRespInterface.end(); ++iterator)
            {
                CHidRespInterface* pInterface = (CHidRespInterface*)*iterator;

                if (buf[1] == 0xc8)
                {
                    pInterface->OnCommandResp(buf);
                }
                else
                {
                    pInterface->OnSensorEvent(buf);
                }
            }
        }
        else
        {
            return -1;
        }
    }
    return 0;
}
```

Here "buf" is the memory to fetch sensor data frame from HY-PCBA-02. Normally it only use 64 bytes for each frame. The second byte is to tell if it's a sensor data frame or command responses.

Commands are sent to HY-PCBA-02 by function "hid_write".

```
int CUsbHidDevice::CommunicateHid(unsigned char* buf)
{
    if (m_pHandle == NULL)
    {
        if (OpenDevice() <= 0)
        {
            return -1;
        }
    }

    int res = hid_write(m_pHandle, buf, 65);
    if (res < 0)
    {
        CloseDevice();
        return -1;
    }
    else
    {
        return 1;
    }
}
```

Function "StartSensor" & "StopSensor" is to turn on/off sensors on PCB.

```
int CUsbHidDevice::StartSensor()
{
    unsigned char buf[256];
    memset(buf, 0, 256);

    buf[0] = 0x1;
    buf[1] = 0x66;
    buf[2] = 0x01;

    return CommunicateHid(buf);
}

int CUsbHidDevice::StopSensor()
{
    unsigned char buf[256];
    memset(buf, 0, 256);

    buf[0] = 0x1;
    buf[1] = 0x66;
    buf[2] = 0x02;

    return CommunicateHid(buf);
}
```

New firmware for sensor fusion

To enable sensor fusion, a new firmware "SensorFusion_20210222.dfu" must be downloaded.