YDLidar-SDK Communication Protocol

Package Format

The response content is the point cloud data scanned by the system. According to the following data format, the data is sent to the external device in hexadecimal to the serial port.
No Intensity Byte Offset:

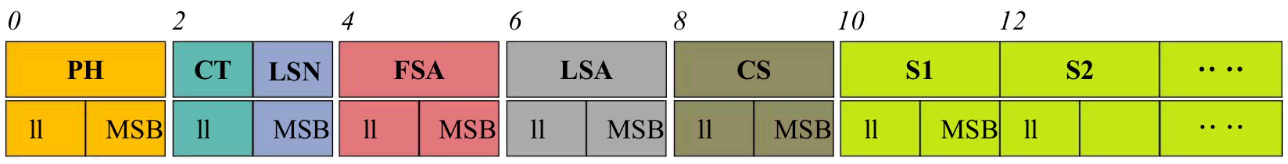

FIG 5 SCAN COMMAND RESPONSE CONTENT DATA STRUCTURE

Scan data format output by LiDAR:

Content	Name	Description
PH(2B)	Packet header	2 Byte in length, Fixed at 0x55AA, low is front, high in back.
CT(1B)	Package type	Indicates the current packet type. (0x00 = CT & 0x01): Normal Point cloud packet. (0x01 = CT & 0x01): Zero packet.
LSN(1B)	Sample Data Number	Indicates the number of sampling points contained in the current packet. There is only once zero point of data in thre zero packet. the value is 1.
FSA(2B)	Starting angle	The angle data corresponding to the first sample point in the smapled data.
LSA(2B)	End angle	The angle data corresponding to the last sample point in the sampled data.
CS(2B)	Check code	The check code of the current data packet uses a two-byte exclusive OR to check the current data packet.
Si(2B)	Sampling data	The system test sampling data is the distance data of the sampling point.

Zero resolution

Start data packet: (CT & 0x01) = 0x01, LSN = 1, Si = 1.
For the analysis of the specific values of distance and angle, see the analysis of distance and angle.

Distance analysis:

- Distance solution formula:
  - Triangle LiDAR:

    ```
    Distance(i) = Si / 4;
    ```

  - TOF LiDAR:

    ```
    Distance(i) = Si;
    ```

Si is sampling data. Sampling data is set to E5 6F. Since the system is in the little-endian mode, the
sampling point S = 0x6FE5, and it is substituted into the distance solution formula, which yields

- Triangle LiDAR:

  ```
  Distance = 7161.25mm
  ```

- TOF LiDAR:

  ```
  Distance = 28645mm
  ```

Angle analysis:

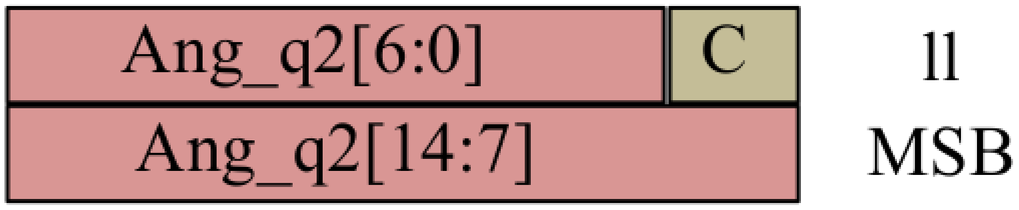

Fig 6 ANGLE

First level analysis:

Starting angle solution formula:

$$Angle_{FSA} = \frac{Rshiftbit(FSA,1)}{64}$$

End angle solution formula:

$$Angle_{LSA} = \frac{Rshiftbit(LSA,1)}{64}$$

Intermediate angle solution formula:

$$Angle_{i} = \frac{diff(Angle)}{LSN-1} * i + Angle_{FSA}(0,1,\ldots,LSN-1)$$

$Angle_{0} : Angle_{FSA};$
$Angle_{LSN-1} : Angle_{LSA};$

`Rshiftbit(data,1)` means shifting the data to the right by one bit.diff Angle means the clockwise angle difference from the starting angle (uncorrected value) to the ending angle (uncorrected value),and LSN represents the number of packet samples in this frame.

`diff(Angle)` :? `(Angle(LSA) - Angle(FSA))` If less than zero, `diff(Angle) = (Angle(LSA)- Angle(FSA)) + 360` , otherwise `diff(Angle) = (Angle(LSA)- Angle(FSA))`

code

```
double Angle_FSA = (FSA >> 1) / 64;
double Angle_LSA = (LSA >> 1) / 64;
double angle_diff = Angle_FSA - Angle_LSA?
if(angle_diff < 0) {
    angle_diff += 360;
}
double Angle[LSN];
for(int i = 0; i < LSN; i++) {
    Angle[i] = i* angle_diff / (LSN - 1) + Angle_FSA;
}
```

Check code parsing:

The check code uses a two-byte exclusive OR to verify the current data packet. The check code itself does not participate in XOR operations, and the XOR order is not strictly in byte order. The XOR sequence is as shown in the figure. Therefore, the check code solution formula is:

$$CS = XOR \sum_{i=1}^{n} (C_i)$$

CS Sequence

PH	**C(1)**
FSA	**C(2)**
S1	**C(3)**
S2	**C(4)**
...	**..**
Sn	**C(n-2)**
[CT | LSN]	**C(n-1)**
LSA	**C(n)**

- Note: XOR(end) indicates the XOR of the element from subscript 1 to end. However, XOR satisfies the exchange law, and the actual solution may not need to follow the XOR sequence.

Code

No intensity Si(2B):

```
uint16_t checksumcal = PH;
checksumcal ^= FSA;
for(int i = 0; i < 2 * LSN; i = i +2 ) {
    checksumcal ^= uint16_t(data[i+1] <<8 | data[i]);
}
checksumcal ^= uint16_t(LSN << 8 | CT);
checksumcal ^= LSA;

## uint16_t : unsigned short
```

example

No Intensity:

Name	Size(Byte)	Value	Contant	Buffer
PH	2	0x55AA	Header	0xAA
				0x55
CT	1	0x01	Type	0x01
LSN	1	0x01	Number	0x01
FSA	2	0xAE53	Starting Angle	0x53
				0xAE
LSA	2	0xAE53	End Andgle	0x53
				0xAE
CS	2	0x54AB	Check code	0xAB

				0x54
S0	2	0x000	0 index Distance	0x00
				0x00

```
uint8_t Buffer[12];
Buffer[0] = 0xAA;
Buffer[1] = 0x55;
Buffer[2] = 0x01;
Buffer[3] = 0x01;
Buffer[4] = 0x53;
Buffer[5] = 0xAE;
Buffer[6] = 0x53;
Buffer[7] = 0xAE;
Buffer[8] = 0xAB;
Buffer[9] = 0x54;
Buffer[10] = 0x00;
Buffer[11] = 0x00;

uint16_t check_code = 0x55AA;
uint8_t CT = Buffer[2] & 0x01;
uin8_t LSN = Buffer[3];
uint16_t FSA = uint16_t(Buffer[5] << 8 | Buffer[4]);
check_code ^= FSA;
uint16_t LSA = uint16_t(Buffer[7] << 8 | Buffer[6]);
uint16_t CS = uint16_t(Buffer[9] << 8 | Buffer[8]);

double Distance[LSN];
for(int i = 0; i < 2 * LSN; i = i + 2) {
    uint16_t data = uint16_t(Buffer[10 + i + 1] << 8 | Buffer[10 + i]);
    check_code ^= data;
    Distance[i / 2 ] = data;
}
check_code ^= uint16_t(LSN << 8 | CT);
check_code ^= LSA;

double Angle[LSN];

if(check_code == CS) {
    double Angle_FSA = (FSA >> 1) / 64;
    double Angle_LSA = (LSA >> 1) / 64;
    double Angle_Diff = (Angle_LSA - Angle_FSA);
    if(Angle_Diff < 0) {
        Angle_Diff = Angle_Diff + 360;
    }
    for(int i = 0; i < LSN; i++) {
        Angle[i] = i * Angle_Diff/ (LSN- 1) + Angle_FSA;
        if( Angle[i] >= 360) {
             Angle[i] -= 360;
        }
    }
}
```

More information, refer to example