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:

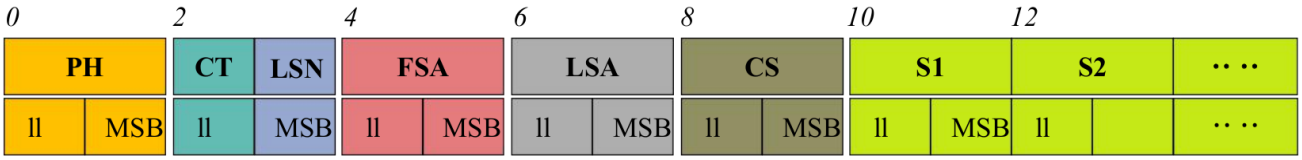

FIG 5 SCAN COMMAND RESPONSE CONTENT DATA STRUCTURE

Intensity Byte Offset:

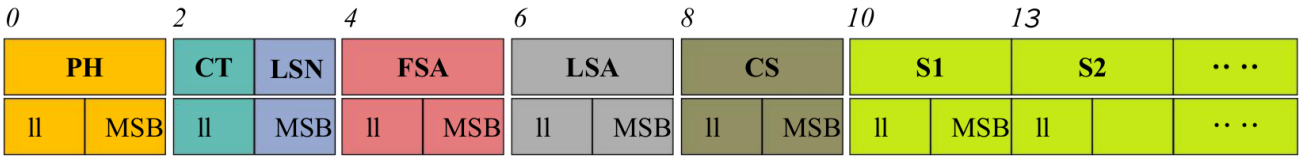

FIG 5 SCAN COMMAND RESPONSE CONTENT DATA STRUCTURE

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/3B)	Sampling data	The system test sampling data is the distance data of the sampling point.

```
Note: If the LiDAR has intensity, Si is 3 Byte. otherwise is 2 Byte.
Si(3B)-->I(1B)(D(2B)): first Byte is Inentsity, The last two bytes are the
Distance.
```

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
```

Intensity analysis:

Si(3B) split into three bytes : S(0) S(1) S(2)

- Inensity solution formula:
  - Triangle LiDAR:

```
Intensity(i) = uint16_t((S(1) & 0x03)<< 8 | S(0));
Distance(i) = uint16_t(S(2) << 8 | S(1)) >> 2;
```

Si is sampling data. Sampling data is set to 1F E5 6F. Since the system is in the little-endian mode, the

- Triangle LiDAR:

```
Intensity = uint16_t((0xE5 & 0x03)<< 8 | 0x1F) = 287;
Distance = uint16_t(0x6F << 8 | 0xE5) >> 2 = 7161mm;
```

Angle analysis:

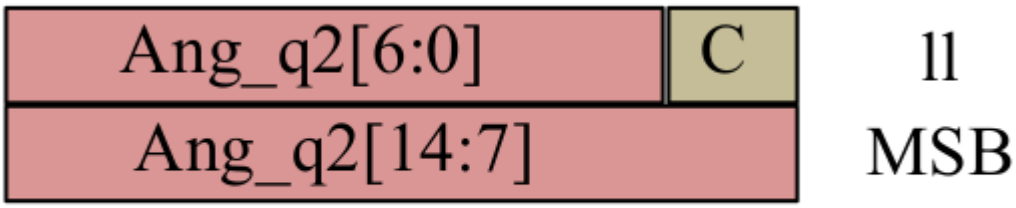

Fig 6 ANGLE

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;
}
```

Second-level analysis:

Triangle Lidar only has current Second-level analysis, TOF Lidar does not need.

Angle correction formula: $Angle_{i} = Angle_{i} + AngCorrect_{i}$; ($1,2,\ldots,LSN$) AngCorrect is the angle correction value, and its calculation formula is as follows, $tand^{-1}$ is an inverse trigonometric function. and the return angle value is:

if($Distance_{i}$ == 0) { $AngCorrect_{i}$ = 0; } else { $AngCorrect_{i} = atan(21.8 * \frac{155.3 - Distance_{i}}{155.3*Distance_{i}})$ }

In the data packet, the 4th to 8th bytes are `28 E5 6F BD 79`, so `LSN = 0x28 = 40(dec)`, `FSA = 0x6FE5`, `LSA = 0x79BD`, and bring in the first-level solution formula, and get: $Angle_{FSA} = 223.78^{°}$ $Angle_{LSA} = 243.47^{°}$ $diff(Angle) = Angle_{LSA} - Angle_{FSA} = 243.47^{°} - 223.78^{°} = 19.69^{°}$ $Angle_{i} = \frac{19.69^{°}}{39}*(i -1) + 223.78^{°}$ ($1,2,\ldots,LSN$) Assume that in the frame data: $Distance_{1} = 1000$ $Distance_{LSN} = 8000$ bring in the second-level solution formula, you get: $AngCorrect_{1} = -6.7622^{°}$ $AngCorrect_{LSN} = -7.8374^{°}$ $Angle_{FSA} = Angle_{1} + AngCorrect_{1} = 217.0178^{°}$ $Angle_{LSA} = Angle_{LSA} + AngCorrect_{LSA} = 235.6326^{°}$ Similarly, $Angle_{i}(2,3, \ldots,LSN-1)$, can be obtained sequentially.

```
for(int i = 0; i < LSN; i++) {
    if(Distance[i] > 0) {
        double AngCorrect = atan(21.8 * (155.3 - Distance[i]) / (155.3 *
Distance[i]));
        Angle[i] += AngCorrect;
    }
    if(Angle[i] >= 360) {
        Angle[i] -= 360;
    }
}
```

Note:

- TOF LiDAR does not neeed second-level analysis.

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
```

Intensity Si(3B):

```
uint16_t checksumcal = PH;
checksumcal ^= FSA;
for(int i = 0; i < 3 * LSN; i = i + 3) {
    checksumcal ^= data[i];
    checksumcal ^= uint16_t(data[i+2] <<8 | data[i + 1]);
}
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 / 4;
}
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(Distance[i] > 0) {
            double AngCorrect = atan(21.8 * (155.3 - Distance[i]) / (155.3
* Distance[i]));
            Angle[i] = Angle[i] + AngCorrect;
        }
        if(Angle[i] >= 360) {
            Angle[i] -= 360;
        }
    }
}
```

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
I0	1	0x00	0 index Intensity	0x00
S0	2	0x000	0 index Distance	0x00
				0x00

```
uint8_t Buffer[13];
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;
Buffer[12] = 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];
uin16_t Itensity[LSN];
for(int i = 0; i < 3 * LSN; i = i + 3) {
    check_code ^= Buffer[10 + i];
    uint16_t data = uint16_t(Buffer[10 + i + 2] << 8 | Buffer[10 + i + 1]);
    check_code ^= data;
    Itensity[i / 3] = uint16_t((Buffer[10 + i + 1] & 0x03) <<8 |  Buffer[10
+ i]);
    Distance[i / 3] = data >> 2;
}
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(Distance[i] > 0) {
            double AngCorrect = atan(21.8 * (155.3 - Distance[i]) / (155.3
* Distance[i]));
            Angle[i] = Angle[i] + AngCorrect;
        }
    }
}
```

For more details and usage examples, Refer to Communication Protocol