Contents

1. WORKING PRINCIPLE ....................................................................................................................................1
   1.1 Communication Interface .....................................................................................................................1
   1.2 Introduction to Functions ......................................................................................................................1

2. COMMUNICATION PROTOCOL ..................................................................................................................3
   2.1 Data Packet Format ...............................................................................................................................3
   2.2 Protocol Command ....................................................................................................................................5
       2.2.1 Command Data Format Remarks ........................................................................................................5
       2.2.2 Start (Mode=0xA0) ...........................................................................................................................6
       2.2.3 Stop (Mode=0xA1) ..............................................................................................................................6
       2.2.4 Speed Control (Mode=0xA2) ..............................................................................................................6
       2.2.5 Query the Current Speed (Mode=0xA3) ............................................................................................7
   2.3 Health Information Data Packet ..............................................................................................................7
       2.3.1 Data Format .......................................................................................................................................7
       2.3.2 Abnormality Remarks .......................................................................................................................8

3. COORDINATE SYSTEM DEFINITION ........................................................................................................10

4. DEVELOPMENT KIT USAGE ........................................................................................................................11
   4.1 Hardware Connection And Remarks ........................................................................................................11
   4.2 Driver Download On Windows ..................................................................................................................12
   4.3 Host Software Usage ................................................................................................................................13
   4.4 SDK Usage on Linux ................................................................................................................................15
       4.4.1 Get SDK Source Code .........................................................................................................................15
       4.4.2 SDK Application Usage .....................................................................................................................15
       4.4.3 SDK Debug Remarks .........................................................................................................................15

5. REVISION ....................................................................................................................................................19
1. WORKING PRINCIPLE

1.1 COMMUNICATION INTERFACE

LD14P connects to external systems via the 1.25mm 4PIN connector for power supply and data reception. Please refer to the following diagram/table for specific interface definitions and parameter:

<table>
<thead>
<tr>
<th>No.</th>
<th>Signal</th>
<th>Type</th>
<th>Remarks</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rx</td>
<td>Input</td>
<td>LiDAR data input</td>
<td>0V</td>
<td>-</td>
<td>3.3V</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
<td>Power</td>
<td>Power negative</td>
<td>-</td>
<td>0V</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Tx.</td>
<td>Output</td>
<td>LiDAR data output</td>
<td>0V</td>
<td>3.3V</td>
<td>3.5V</td>
</tr>
<tr>
<td>4</td>
<td>VCC</td>
<td>Power</td>
<td>Power positive</td>
<td>4.5V</td>
<td>5V</td>
<td>5.5V</td>
</tr>
</tbody>
</table>

1.2 INTRODUCTION TO FUNCTIONS

Adopting triangulation technology, the LD14P LiDAR measures distance at a rate of 4000 times per second. During each measurement, LD14P emits infrared laser from a fixed angle, which is then reflected by the target object and received by the receiving unit. By establishing a triangle relationship among the laser, the target object, and the receiving unit, the distance is calculated.

After obtaining distance data, LD14P merges the angle values measured by the angle measurement unit into point cloud data. This data is then wirelessly transmitted to an external
interface.

LD14P supports both internal and external speed control. When the Rx pin of LD14P is pulled down to ground, upon power-up, LD14P enters the internal velocity control mode by default, with a scanning frequency of 6Hz. The external speed control supports both PWM control and serial port control.

1) PWM control:

Connect the Rx pin of LD14P to PWM signal, which can be used to control the start, stop and speed of the motor via the PWM signal duty ratio for activating external speed control.

a. Input PWM frequency of 500Hz-1.5KHz, 1KHz is recommended.

b. Duty ratio in the range of (45%, 55%) (excluding 45% and 55%), requires a minimum input duration of 100ms.

Once triggered, the LD14P remains in external speed control mode unless it is powered off and rebooted to internal speed control. In addition, speed control is available by adjusting the PWM duty ratio. Due to individual differences in each product motor, the actual speed may vary when the duty ratio is set to typical values. For precise control of the motor speed, closed-loop control is required based on the speed information in the received data.

Note: When external speed control is not used, the PWM pin must be grounded.

2) Serial port control:

The Rx pin of LD14P connects to the Tx pin of an external device's serial port. To enter external speed control, relevant control commands should be sent at the baud rate defined by the communication interface.
2. COMMUNICATION PROTOCOL

2.1 DATA PACKET FORMAT

<table>
<thead>
<tr>
<th>Start Character</th>
<th>Frame</th>
<th>LiDAR Speed</th>
<th>Start Angle</th>
<th>Measurement Data</th>
<th>End Angle</th>
<th>Timestamp</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>1 Byte</td>
<td>2 Bytes</td>
<td>2 Bytes</td>
<td>12x3 Bytes</td>
<td>2 Bytes</td>
<td>2 Bytes</td>
<td>1 Byte</td>
</tr>
<tr>
<td>0x54</td>
<td>0x2C</td>
<td>LSB</td>
<td>MSB</td>
<td></td>
<td>LSB</td>
<td>MSB</td>
<td></td>
</tr>
</tbody>
</table>

- **Start character:** 1 byte, fixed value of 0x54, indicating the start of the data packet.
- **Frame:** 1 byte, fixed value of 0x2C, the lower five bits indicate the number of measurement points in a packet, currently fixed at 12.
- **LiDAR speed:** 2 bytes, unit in degree per second.
- **Start angle:** 2 bytes, unit in 0.01 degrees.
- **Measurement data:** 3 bytes, consisting of LSB distance, MSB distance and intensity information.
- **End angle:** 2 bytes, unit in 0.01 degrees.
- **Timestamp:** 2 bytes, unit in milliseconds, up to 30000, reset upon reaching 30000.
- **CRC checksum:** CRC8 checksum for all preceding data.

Data structure reference as follows:

```c
#define POINT_PER_PACK 12
#define HEADER 0x54
typedef struct __attribute__((packed)) {
    uint16_t distance;
    int8_t intensity;
} LidarPointStructDef;
typedef struct __attribute__((packed)) {
    uint8_t header;
    uint8_t ver_len;
    uint16_t speed;
    uint16_t start_angle;
} LidarPointStructDef point[POINT_PER_PACK];
```
CRC checksum calculation as follows:

```c
static const uint8_t CrcTable[256] =
{
  0x00, 0x4d, 0x9a, 0xd7, 0x79, 0x34, 0xe3,
  0xae, 0xf2, 0xbf, 0x68, 0x25, 0x8b, 0xc6, 0x11, 0x5c, 0xa9, 0xe4, 0x33,
  0x7e, 0xd0, 0x9d, 0x4a, 0x07, 0x5b, 0x16, 0xc1, 0x8c, 0x22, 0x6f, 0xb8,
  0xf5, 0x1f, 0x52, 0x85, 0xc8, 0x66, 0x2b, 0xfc, 0xb1, 0xed, 0xa0, 0x77,
  0x3a, 0x94, 0xd9, 0xe0, 0x43, 0xb6, 0xf8, 0xe2, 0x61, 0xcf, 0xb2, 0x82,
  0x18, 0x44, 0x09, 0xde, 0x93, 0x3d, 0x70, 0xe7, 0xe9, 0x3e, 0x73, 0xa4,
  0xe9, 0x47, 0xe0, 0xd0, 0x90, 0xc0, 0x81, 0xc6, 0x1b, 0xb5, 0xf8, 0x2f,
  0x62, 0x97, 0xda, 0x0d, 0x40, 0xce, 0xa3, 0x74, 0x39, 0x65, 0xc2, 0x78,
  0xb2, 0x1c, 0x51, 0x86, 0xcb, 0x21, 0x6c, 0xb0, 0xf6, 0x58, 0x15, 0xc2,
  0x8f, 0xd3, 0x9e, 0x49, 0x04, 0xa1, 0xe7, 0x30, 0xe7, 0x88, 0xc5, 0x12,
  0x5f, 0xf1, 0xbc, 0x6b, 0x26, 0x7a, 0xe3, 0xe0, 0xad, 0x03, 0x4e, 0x99,
  0xd4, 0x7c, 0x31, 0xe6, 0xab, 0x05, 0x48, 0x9f, 0xd2, 0xe8, 0xc3, 0x14,
  0x59, 0xf7, 0xba, 0x6d, 0x70, 0x5d, 0x98, 0x4f, 0x92, 0xa1, 0xe1, 0x35,
  0x7b, 0x27, 0x6a, 0xbd, 0x70, 0x5e, 0x12, 0xc4, 0x89, 0x63, 0x2e, 0xf9,
  0xb4, 0x1a, 0x57, 0x80, 0xcd, 0x91, 0xdc, 0x0b, 0x46, 0xe8, 0xa5, 0x72,
  0x3f, 0xca, 0x87, 0x50, 0x1d, 0xb3, 0xf2, 0x29, 0xd4, 0x38, 0x75, 0xa2,
  0xef, 0x41, 0x0c, 0xdb, 0x96, 0x42, 0x8f, 0xd8, 0x95, 0x3b, 0x76, 0x1a,
  0xec, 0x80, 0xfd, 0x2a, 0x67, 0xc9, 0x84, 0x53, 0x1e, 0xeb, 0xa6, 0x71,
  0x3c, 0x92, 0xdf, 0x08, 0x45, 0x19, 0x54, 0x83, 0xc6, 0x60, 0x2d, 0xfa,
  0xb7, 0x5d, 0x10, 0xc7, 0x8a, 0x24, 0x69, 0xbe, 0xf3, 0xaf, 0xe2, 0x35,
  0x78, 0xd6, 0x9b, 0xc4, 0x01, 0xf4, 0xb9, 0xe6, 0x23, 0x8d, 0xc0, 0x17,
  0x5a, 0x06, 0x4b, 0x9c, 0xd1, 0x7f, 0x32, 0xe5, 0xa8
};

uint8_t CalCRCB(uint8_t *p, uint8_t len)
{
  uint8_t crc = 0;
  uint16_t i;
  for (i = 0; i < len; i++)
  {
    crc = CrcTable[(crc ^ *p++) & 0xff];
  }
  return crc;
}
2.2 PROTOCOL COMMAND

2.2.1 COMMAND DATA FORMAT REMARKS

Command data remarks:

<table>
<thead>
<tr>
<th>Start Character</th>
<th>Command Code</th>
<th>Data Length</th>
<th>Command Data</th>
<th>Parity Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>1 Byte</td>
<td>1 Byte</td>
<td>4 Bytes</td>
<td>1 Byte</td>
</tr>
<tr>
<td>0x54</td>
<td>Mode</td>
<td>0x04</td>
<td>DataBuffer</td>
<td>CRC8</td>
</tr>
</tbody>
</table>

- **Start character**: 1 byte, fixed value of 0x54, indicating the start of the data packet.
- **Command code**: 1 byte, command mode.
- **Data length**: data length, fixed at 0x04.
- **Command data**: command data.
- **CRC checksum**: CRC8 checksum for all preceding data.

Returned data remarks:

<table>
<thead>
<tr>
<th>Start Character</th>
<th>Command Code</th>
<th>Data Length</th>
<th>Command Data</th>
<th>Parity Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>1 Byte</td>
<td>1 Byte</td>
<td>4 Bytes</td>
<td>1 Byte</td>
</tr>
<tr>
<td>0x54</td>
<td>Mode</td>
<td>0x04</td>
<td>DataBuffer</td>
<td>CRC8</td>
</tr>
</tbody>
</table>

- **Start character**: 1 byte, fixed value of 0x54, indicating the start of the data packet.
- **Command code**: 1 byte, receive the command mode.
- **Data length**: the minimum data length is 4 bytes, depending on the specific command.
- **Command data**: response data, the length varies depending on the commands.
- **CRC checksum**: CRC8 checksum for all preceding data.

Function summary:

<table>
<thead>
<tr>
<th>Command Code</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA0</td>
<td>Start the LiDAR motor</td>
</tr>
<tr>
<td>0xA1</td>
<td>Stop the LiDAR motor</td>
</tr>
<tr>
<td>0xA2</td>
<td>Set the target speed of LiDAR, data lose</td>
</tr>
<tr>
<td></td>
<td>when power off</td>
</tr>
<tr>
<td>0xA3</td>
<td>Query the LiDAR target speed</td>
</tr>
</tbody>
</table>
2.2.2 START (MODE=0xA0)

When Mode=0xA0, the command is to start the motor, and the DataBuffer in the command has no actual meaning and defaults to 0. The DataBuffer length in the LiDAR response is 4 bytes and has no actual meaning.

Reference command sent: 0x54 A0 04 00 00 00 00 5E
Reference command received: 0x54 A0 04 00 00 00 00 5E

2.2.3 STOP (MODE=0xA1)

When Mode=0xA1, the command is to stop the motor, and the DataBuffer in the command has no actual meaning and defaults to 0. The DataBuffer length in the LiDAR response is 4 bytes and has no actual meaning.

Reference command sent: 0x54 A1 04 00 00 00 00 4A
Reference command received: 0x54 A1 04 00 00 00 00 4A

2.2.4 SPEED CONTROL (MODE=0xA2)

When Mode=0xA2 in the command, the function is to set the speed, and the DataBuffer in the command represents the target speed. The DataBuffer length in the LiDAR response is 4 bytes, indicating the received target speed from the command.

Speed analysis:

\[
\text{speed} = (\text{DataBuffer}[0]) | (\text{DataBuffer}[1] << 8);
\]

Speed unit: degrees/second

Reference command sent: 0x54 A2 04 D8 09 00 00 16 (setting speed to 2520 degrees/second, i.e., 7Hz)

Reference response received: 0x54 A2 04 D8 09 00 00 16

Reference command sent: 0x54 A2 04 70 08 00 00 A1 (setting speed to 2160 degrees/second, i.e., 6Hz)

Reference response received: 0x54 A2 04 70 08 00 00 A1
2.2.5 QUERY THE CURRENT SPEED (MODE=0XA3)

When Mode=0xA3 in the command, the function is to query the current target speed, and the DataBuffer in the command has no actual meaning. The DataBuffer length in the LiDAR response is 4 bytes, indicating the current target speed of the LiDAR.

Speed analysis:

\[
\text{speed} = (\text{DataBuffer}[0]) | (\text{DataBuffer}[1] << 8);
\]

Speed unit: degrees/second

Reference command sent: 0x54 A3 04 00 00 00 00 62

Reference response received: 0x54 A3 04 D8 09 00 00 02 (setting the target speed to 2520 degrees/second, i.e., 7Hz)

2.3 HEALTH INFORMATION DATA PACKET

The LiDAR has an error reporting mechanism where it reports health information, including error codes, at intervals of every 1 second.

2.3.1 DATA FORMAT

<table>
<thead>
<tr>
<th>Start Character</th>
<th>Information Remarks</th>
<th>Error Code</th>
<th>Error Mode</th>
<th>Parity Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>1 Byte</td>
<td>2 Bytes</td>
<td>1 Byte</td>
<td>1 Byte</td>
</tr>
<tr>
<td>0x54</td>
<td>Information</td>
<td>ErrorMode</td>
<td>ErrorMode</td>
<td>CRC8</td>
</tr>
</tbody>
</table>

- **Start character**: 1 byte, fixed value of 0x54, indicating the start of the data packet.
- **Information remarks**: 1 byte, fixed at 0x1F.
- **Error code**: 2 bytes, as shown below:
<table>
<thead>
<tr>
<th>Bit0~Bit15</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit0</td>
<td>Laser tube abnormal</td>
</tr>
<tr>
<td>Bit1</td>
<td>Laser tube abnormal</td>
</tr>
<tr>
<td>Bit2</td>
<td>Receiver board voltage abnormal</td>
</tr>
<tr>
<td>Bit3</td>
<td>Sensor abnormal</td>
</tr>
<tr>
<td>Bit4</td>
<td>Encoder abnormal</td>
</tr>
<tr>
<td>Bit5</td>
<td>Motor error or locked rotor</td>
</tr>
<tr>
<td>Bit6</td>
<td>LiDAR blockage</td>
</tr>
<tr>
<td>Bit7~Bit15</td>
<td>TBD</td>
</tr>
</tbody>
</table>

- **Error mode**: error mode, 0x00: normal, 0x01: warn, 0x02: error
- **CRC checksum**: CRC8 checksum for all preceding data.

### 2.3.2 ABNORMALITY REMARKS

1) Laser tube abnormal: Check whether the laser tube is abnormal by the collected voltage when it is turned on, which occurs only once during power-on self-test. The possible causes of abnormality include laser tube damage or improper connection between the laser tube and the received board.

2) Receiver board voltage abnormal: Check whether the receiver board voltage is abnormal by the collected power voltage, which occurs only once during power-on self-test. The possible causes of this abnormality include coil issues or the abnormal power supply of the transmission board.

3) Sensor abnormal: The sensor is abnormal when its timing is irregular. The possible causes of this abnormality may be the sensor damage or the improper sensor connection to the receiver board.

4) Encoder abnormal: Check whether the encoder is abnormal by counting the total number of teeth in one rotation. If the number of teeth in one rotation does not match the actual numbers, it’s abnormal. Possible reasons for this abnormality may be encoder or encoder disc damage.

5) Motor abnormal or locked rotor: Check whether the motor is blocked or not by judging the
accumulated number of points in one rotation. If the accumulated points exceed 4000 points in one rotation, it is abnormal. Possible causes include motor locked rotor, motor damage or encoder abnormal.

6) LiDAR blockage: if there is no effective data in one rotation, the LiDAR is blocked.
3. COORDINATE SYSTEM DEFINITION

The LD14P commonly follows a left-hand rule coordinate system where the front of the sensor is defined as the X-axis of the coordinate system (i.e. the 0-angle position), the origin of the coordinate system is the center of rotation of the ranging unit, and the angle of rotation increases along the clockwise direction, as shown in the following diagram:
4. DEVELOPMENT KIT USAGE

4.1 HARDWARE CONNECTION AND REMARKS

1) LD14P LiDAR, wires, and CP2102 USB TO TTL Module

2) The connection is shown below:
4.2 DRIVER DOWNLOAD ON WINDOWS

When evaluating our company’s product on Windows, it's necessary to install the USB-to-serial driver for the USB adapter. This is because the USB adapter used in our development kit employs the CP2102 USB-to-serial signal chip, and its driver demo can be downloaded from Silicon Labs' official website.

After extracting the CP210x_Universal_Windows_Driver driver package, run the .exe file in the driver installation package directory, choosing between X86 (32-bit) or X64 (64-bit) based on the Windows system version.

Double-click the .exe file and install it by prompts:

![CP210x USB to UART Bridge Driver Installer](image)
After the installation is complete, connect the USB adapter from the development kit to your computer. Right-click on "My Computer," select "Properties," navigate to the "Device Manager" in the "System" interface that opens up, expand the "Ports" section, and there you'll find the assigned COM port number corresponding to the recognized CP2102 USB adapter. This indicates a successful driver installation. COM4 is shown in the image below:

![Device Manager Image]

4.3 HOST SOFTWARE USAGE

Our company provides the LdsPointCloudViewer software for real-time visualization of point clouds generated by our product. Developers can use this software to intuitively observe the scanning results of our product.

Before using this software, ensure that the USB driver for our product's USB adapter has been successfully installed. Connect our product to a USB port on a Windows PC, then double-click on
LdsPointCloudViewer.exe. Select the corresponding product model (e.g., LD14P) and port number. Click the 'Start Point Cloud Refresh' button as shown in the image below:

In the above picture, "Speed" indicates the scanning frequency of the LiDAR in Hertz (Hz). "Rate" represents the data packet parsing speed of the LiDAR. "Valid" indicates the number of valid points measured by the LiDAR in one full rotation.

The binary package of the LdsPointCloudViewer software can be accessed for download from our company's open-source repository:

https://github.com/LdrobotSensorTeam/ld_desktop_tool/releases
4.4 SDK USAGE ON LINUX

4.4.1 GET SDK SOURCE CODE

Beta Version: The Linux SDK source code for this product requires contacting our company's FAE or other sales for access. The README document within the source code contains relevant usage instructions. If you wish to utilize a private repository on platforms like GitHub or Gitee for development integration, please feel free to inform our company's FAE or sales. We will strive to meet your development needs.

Official Version: The product's SDK is not currently available to the public.

4.4.2 SDK APPLICATION USAGE

To facilitate users to connect this product to the sensor, this product's SDK integrates fundamental functionalities such as data parsing, processing, and packaging. It's compatible with various commonly used development platforms (Linux, ROS, ROS2). Currently, the integrated functionalities within the SDK are as follows:

1. Data serial port reception buffer addressing
2. Data packet protocol parsing and processing
3. Data noise filtering
4. Data packaging and provision of corresponding interfaces to upper layers.

4.4.3 SDK DEBUG REMARKS

1) SDK platform

This demo is based on ROS platform on Ubuntu operating system, please install the correct Ubuntu and ROS environment following the example operation. If you intend to evaluate this product using SDK on other platforms, you can refer to the corresponding platform README document of the SDK source code.

2) Set up sensor permissions

Firstly, connect the LiDAR to the computer via a serial-to-USB module. Then, in the Ubuntu
system, open the terminal and inputs `ls /dev/ttyUSB*` to check if the serial port is connected. If the serial device is detected, use the command `sudo chmod 777 /dev/ttyUSB0` to grant it the highest permissions. This command grants read, write, and execute permissions to the file owner, group, and other users.

```bash
ls /dev/ttyUSB*
sudo chmod 777 /dev/ttyUSB0
```

3) Compile remarks

```bash
$ cd <sdk software package root directory>
$ catkin_make
```

Successful compilation will result in the following interface:

```
Scanning dependencies of target ldflidar
13% Building CXX object ld14/CMakeFiles/ld14_lidar.dir/src/main.cpp.o
13% Building CXX object ld14/CMakeFiles/ld14_lidar.dir/src/roslidar_drivers/loggen/bug.c.o
33% Building CXX object ld14/CMakeFiles/ld14_lidar.dir/src/ld14_liquidator.cpp.o
44% Building CXX object ld14/CMakeFiles/ld14_lidar.dir/src/ld14_lidar_drivers/loggen/bug.c.o
55% Building CXX object ld14/CMakeFiles/ld14_lidar.dir/src/ld14_lidar_drivers/loggen/bug.c.o
66% Building CXX object ld14/CMakeFiles/ld14_lidar.dir/src/ld14_lidar_drivers/loggen/bug.c.o
77% Building CXX object ld14/CMakeFiles/ld14_lidar.dir/src/ld14_lidar_drivers/loggen/bug.c.o
88% Building CXX object ld14/CMakeFiles/ld14_lidar.dir/src/ld14_lidar_drivers/loggen/bug.c.o
100% Linking CXX executable ld14_lidar/ld14_binary_file_name.o
100% multi_target ld14_lidar
```

4) Run the corresponding node:

After the compilation is complete, it's necessary to add the compiled files to the environment variables using the command `source devel/setup.bash`. This command temporarily adds the environment variables to the terminal. This means that if you open a new terminal, you'll also need to navigate to the 'sdk ld14' path and execute the command to add the environment variables.

Once the environment variables are added, the 'roslaunch' command will be able to locate the respective ROS package and launch files. Run the command `roslaunch ld14_lidar ld14.launch`. 

---

16
The operation interface is shown below:

```bash
$ cd <sdk software package root directory>
$ source devel/setup.bash
# Start the device node
$ roslaunch lididar ld14.launch
# or
# Start and display data on Rviz, suitable for ubuntu20.04 noetic
$ roslaunch lididar viewer_ld14_noetic.launch
# Start and display data on Rviz, suitable for ubuntu16.04 kinetic
# ubuntu18.04melodic$ source devel/setup.bash
$ roslaunch lididar viewer_ld14_kinetic_melodic.launch#
```

5) Rviz display remarks:
Rviz is a commonly used 3D visualization tool in ROS, and supports displaying the LiDAR data. After successfully running roslaunch, open a new terminal and type `rosrun rviz rviz`. Then, click on file -> open -> Config, and then select `sdk_ld14_ros/rviz/lididar.rviz` file.
## 5. REVISION

<table>
<thead>
<tr>
<th>Version</th>
<th>Revision Date</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2022-10-19</td>
<td>Initial creation</td>
</tr>
<tr>
<td>0.2</td>
<td>2023-01-11</td>
<td>Reformatting, optimized relevant descriptions</td>
</tr>
</tbody>
</table>