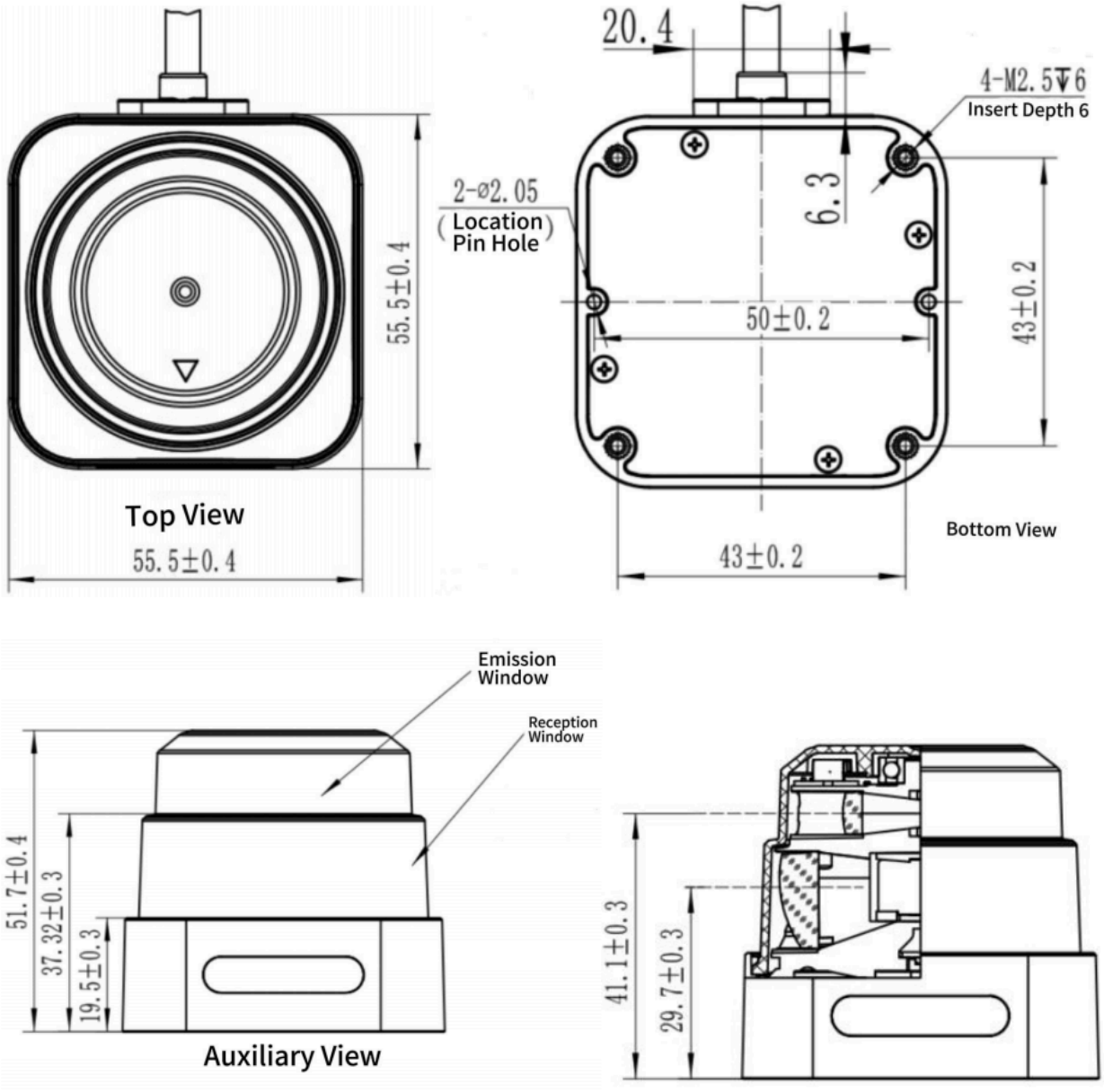
RobiLidar-2D-01 User Manual

Mechanical Dimensions & Optical Window

Mechanical Dimensions

Unit：mm

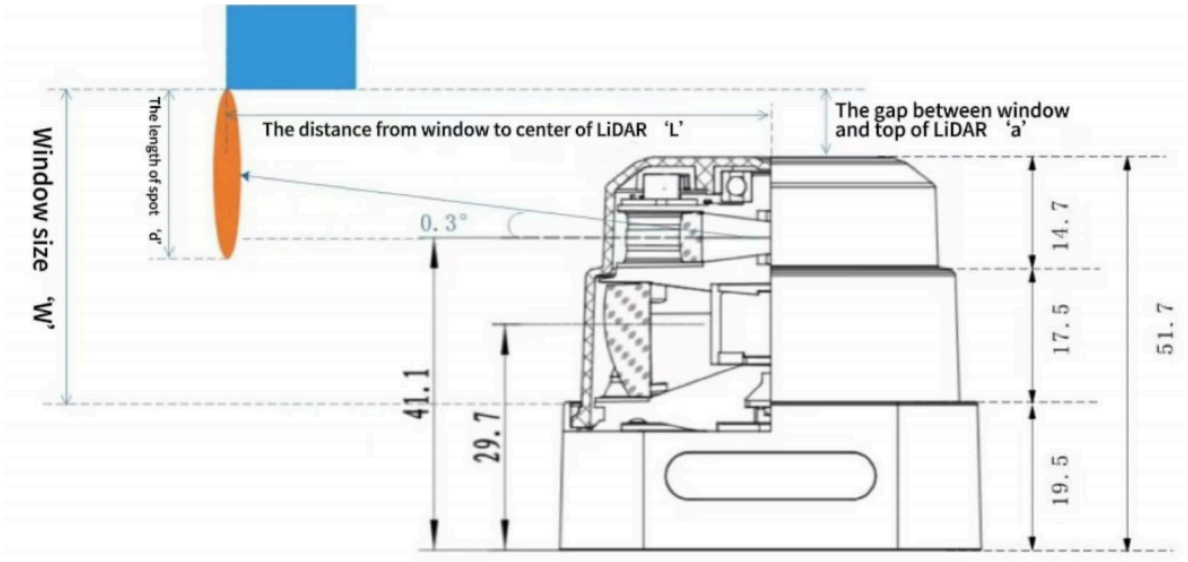


Optical Windows

As shown in Fig., LiDAR installation angle, spot pitch angle and spot length need to be considered when designing the device window. The center of the default device window coincides with the optical axis of the LiDAR transmitting and receiving system. Assuming that the assembly Angle error is 1° and the LiDAR pitch Angle is 0.3°, the distance from window to

center of LiDAR is L=0.5m.The length of spot d=(6.5+5.2*L)/1000. The length of spot at 0.5m is 9.1mm. The gap between the window and the top of the LiDAR a=L*Tan(1+0.3) +0.5d-10.6, the calculated gap distance is 5.30mm; Window size W=32.2+a.According to the calculation, the window size is at least 37.5mm.

Regardless of whether the LiDAR pitch Angle and the machine assembly Angle are positive or negative, the overall window size needs to ensure that the emission and reception window are not blocked.

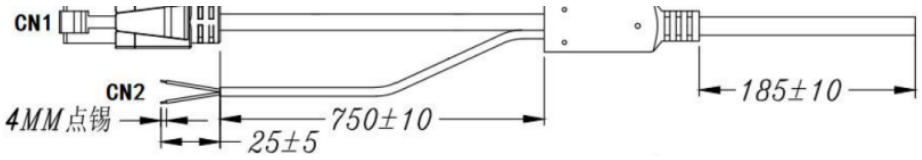


Parameter Performance

Laser Wavelength	905nm±15nm
Ranging Distance	0.1-40m@90% reflection
	0.1-15m@10% reflection
Laser divergence angle	<5.2 mrad
Laser horizontal parallelism	±0.3°
Rotation direction	anticlockwise
Scanning area	360°
Scanning frequency	10Hz,15Hz（600RPM,900RPM,with a fluctuation range of ±5%）
Angle resolution	0.1°,0.15°
Ranging speed	36000Measurement value/s
Ranging accuracy	±25mm（The ranging accuracy is ±25mm under the reflection condition of 10%~ 90% within 7m, and the reliability is 90%）
Distance resolution	10mm
Light spot	Vertical length:6.5mm+4.3×distance(mm) Horizontal length:13mm+0.9×distance(mm)
Port	Ethernet UDP 100M full duplex
Power	<2.5W
Output	Original data（Distance, Angle, Energy）
Ambient light	＞80000lux
Power supply	10~26VDC
Working temperature	-10°C~50°C
Storage dimensions	-25°C~65°C
Size	55.5mm*55.5mm*51.7mm (Length * width * height）

Interface Definition

1. Interface and Reading Guide



Wiring Method		
CN1	Color	Signal Definition
1	Orange - white	TX+
2	Orange	TX-
3	Blue - white	RX+
6	Blue	RX-
CN2	Color	Signal Definition
1	Red	24V
2	Black	GND

The cable drawing and wiring table above complete the Interface Definition that begins on the preceding page. CN1 carries Ethernet TX+/TX-/RX+/RX-; CN2 carries power and ground. Follow the labeled connector pin numbers and wire colors in the drawing.

Pages	Subject
1-3	Mechanical dimensions, optical window design and original performance specifications
4-6	Electrical interface, interpretation notes and network setup
7-9	LiDA device discovery heartbeat and a minimal receiver
10-13	C7FA scan packet format, sample, checksum and assembly
14-15	LMSG alarm format and original example
16-19	Commands, CRC, verified queries, keepalive and timestamps
20	SDK integration and troubleshooting

Model and verification scope

Behavior marked observed was verified on one device reporting LDS-50C-E-R, serial LH1262260500221, firmware raw value 0x12d3acb7, on 27 October 2026. Its mapping to every RobiLidar-2D-01 variant has not been established. Generic SDK interfaces and historical examples do not guarantee support on every unit.

The original hardware drawings and specifications on pages 1-3 are preserved. The protocol clarifications and corrections in the following sections govern software interpretation.

2. Specifications and Network Setup

Interpretation of the original specifications

The original specification table lists 10 Hz / 15 Hz (600 / 900 RPM), with a fluctuation range of +/-5%, and angular resolutions 0.1 degrees / 0.15 degrees. Other official introductory information and the observed approximately 2,000-point revolution at 14.70 Hz imply approximately 0.18 degrees. These figures are inconsistent across sources. Confirm the required resolution and operating mode for the supplied unit; do not assume 0.15-degree output at 15 Hz.

The original optical-window calculation on pages 1-2 is retained: L = 0.5 m, spot length 9.1 mm, calculated clearance 5.30 mm and minimum window size 37.5 mm. The displayed formula uses mixed units; apply consistent units in an engineering calculation. Keep both emission and reception paths unobstructed for either sign of pitch and assembly error.

Factory network configuration

Setting	Factory value
Device IPv4 address	192.168.158.98
Subnet mask	255.255.255.0
Gateway	192.168.158.1
Scan/alarm upload destination	192.168.158.15, UDP port 6668
Device command service	UDP port 6543

First connection

1. Connect Ethernet and the specified DC supply using the interface pinout. Set the host adapter connected to the lidar to 192.168.158.15/24, avoiding an address conflict.

2. Bind the scan receiver to UDP 6668 and allow incoming traffic through the firewall. The upload destination must match the host address that actually receives the data.

3. Capture packets on that adapter. Check source, destination and port separately. Sharing the subnet is insufficient when the lidar uploads to a different IP address.

4. To use another host address, set the lidar upload destination through the supported configuration interface or LSDST command, then verify the actual destination in traffic.

3. Network Channels and Ports

Discovery, scan upload, command exchange and host keepalive are separate channels. Receiving one does not prove that the others are configured correctly.

Channel	Observed / documented path	Purpose
Device discovery heartbeat	192.168.158.98:6545 -> 255.255.255.255:6789 LiDA, 112 bytes, approximately once per second	Identity and network/status advertisement
Scan and alarm upload	192.168.158.98:6544 -> 192.168.158.15:6668 C7FA scans and LMSG alarms	Measurement data and alarm/status messages
Host command / response	192.168.158.15:6668 -> 192.168.158.98:6543 Request 0x0043; response 0xffbc for verified queries	Read identity and issue supported controls
Host keepalive / response	Host -> device:6543 Request 0x4b41; observed response 0xb4be; each has 32-byte payload	Separate keepalive and timing exchange

Traffic diagram

```
Lidar 192.168.158.98                Host 192.168.158.15
  :6545 -- LiDA discovery ----------> broadcast :6789
  :6544 -- C7FA scans / LMSG -------> host      :6668
  :6543 <--- command / keepalive ---- host      :6668
  :6543 ---- replies ---------------> host      :6668
```

Broadcast and multicast are different

The SDK and ROS2 discovery receiver bind 0.0.0.0:6789 and join 225.225.225.225. This receiver configuration is not proof that a device transmits multicast. The tested device sent LiDA packets to broadcast destination 255.255.255.255. Verify the actual destination IP in a packet capture. An IGMP membership report shows group membership, not that the lidar sends data to that group.

On a computer with multiple adapters, select the adapter connected to the lidar for routing, multicast membership and capture. Missing discovery alone does not prove the lidar is offline; check scan upload and command replies separately.

4. LiDA Discovery Heartbeat

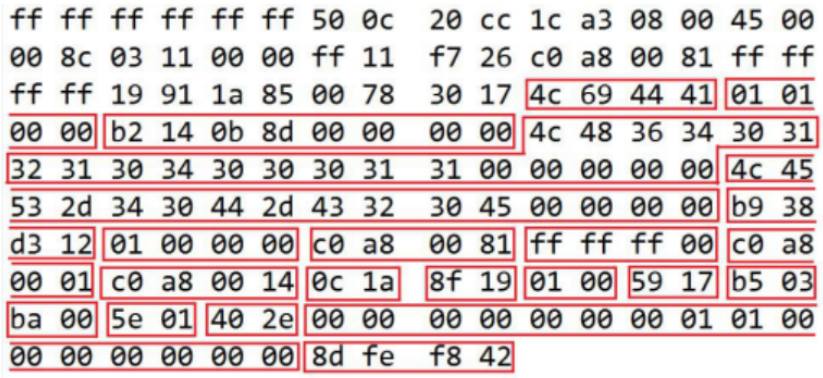
Multibyte numeric fields are little-endian. This layout totals 112 bytes. The two timestamp words remain opaque uint32 values; do not assume they form a 64-bit Unix timestamp.

Offset	Field	Type	Meaning
0	sign	char[4]	ASCII LiDA (4c 69 44 41)
4	proto_version	uint32	Protocol version
8	timestamp / headtime	uint32[2]	Two words; time interpretation firmware-dependent
16	dev_sn	char[20]	Device serial string
36	dev_type	char[16]	Device model string
52	version	uint32	Program/firmware version
56	dev_id	uint32	Device ID
60	ip	uint8[4]	Device IPv4 address
64	mask	uint8[4]	Subnet mask
68	gateway	uint8[4]	Gateway
72	remote_ip	uint8[4]	Upload destination IPv4 address
76	remote_udp	uint16	Upload destination port
78	port	uint16	Device service port
80	status	uint16	Device status; firmware-specific meaning
82	rpm	uint16	Value / 10 = RPM; 6000 means 600 RPM
84	freq	uint16	Value / 100 = Hz; 1000 means 10 Hz
86	ranger_version	uint8[2]	Ranging-head version bytes
88	CpuTemp	int16	Signed value / 10 = degrees C; 270 means 27 C
90	InputVolt	uint16	Value / 1000 = V; 12000 means 12 V
92	alarm	uint8[16]	Alarm information; preserve raw bytes
108	crc	uint32	Little-endian check code

Corrections: CpuTemp is temperature, not CPU voltage, and is signed int16 in the SDK. Do not invent meanings for status/alarm bits. On the observed 112-byte V101 heartbeat, the final CRC32 matched the command STM32 algorithm over the first 108 bytes (Section 13).

5. Original Discovery Example

This historical LES-40D-C20E example uses a 192.168.0.x network. It is retained for field interpretation; it is not the factory configuration or identity of the tested LDS-50C-E-R.



Network Heartbeat Packet Analysis

Bytes	Interpretation
4c 69 44 41 / 01 01 00 00	ASCII LiDA / protocol version 0x00000101
b2 14 0b 8d 00 00 00 00	Timestamp words 0x8d0b14b2 and 0; originally displayed together in ms, without a verified epoch definition
4c 48 36 34 30 31 32 32 30 34 30 30 30 30 31 00 00 00 00 00	Serial LH6401220400001
4c 45 53 2d 34 30 44 2d 43 32 30 45 00 00 00 00	Model LES-40D-C20E
b9 38 d3 12 / 01 00 00 00	Firmware 0x12d338b9 / device ID 1
c0 a8 00 81 / ff ff ff 00	Device 192.168.0.129 / mask 255.255.255.0
c0 a8 00 01 / c0 a8 00 14	Gateway 192.168.0.1 / destination 192.168.0.20
0c 1a / 8f 19 / 01 00	Upload port 6668 / service port 6543 / status 0x0001
59 17 / b5 03	0x1759 / 10 = 597.7 RPM; 0x03b5 / 100 = 9.49 Hz
ba 00 / 5e 01 / 40 2e	Ranging version 0x00ba / 35.0 C / 11.840 V
00 00 00 00 00 00 00 01 01 00 00 00 00 00 00 00	Alarm bytes; retain without invented meanings
8d fe f8 42	Check code

Arithmetic correction: 0x03b5 = 949, so the frequency is 9.49 Hz. The former 94.9 Hz label was incorrect.

6. Minimal Discovery Receiver

Python 3: select the IPv4 address of the Ethernet adapter connected to the lidar. Binding 0.0.0.0:6789 accepts incoming discovery traffic. Optional group membership is selected on that adapter; it does not convert broadcast into multicast.

```
import socket
import struct

INTERFACE_IP = "192.168.158.15"
JOIN_MULTICAST = False
GROUP = "225.225.225.225"
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
s.bind(("0.0.0.0", 6789))
if JOIN_MULTICAST:
    membership = (socket.inet_aton(GROUP) +
                  socket.inet_aton(INTERFACE_IP))
    s.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP,
                 membership)
while True:
    data, source = s.recvfrom(65535)
    if len(data) != 112 or data[:4] != b"LiDA":
        continue
    version = struct.unpack_from("<I", data, 4)[0]
    serial = data[16:36].split(b"\0", 1)[0].decode("ascii", "replace")
    model = data[36:52].split(b"\0", 1)[0].decode("ascii", "replace")
    ip = socket.inet_ntoa(data[60:64])
    destination = socket.inet_ntoa(data[72:76])
    upload_port, service_port = struct.unpack_from("<HH", data, 76)
    rpm_raw, freq_raw = struct.unpack_from("<HH", data, 82)
    temp_raw, volts_raw = struct.unpack_from("<hH", data, 88)
    print(source, serial, model, "protocol", hex(version))
    print("device", ip, "service", service_port,
          "upload", destination, upload_port)
    print("rpm", rpm_raw / 10, "Hz", freq_raw / 100,
          "C", temp_raw / 10, "V", volts_raw / 1000)
```

recvfrom() returns the source address only, not the destination IP. To prove broadcast versus multicast, inspect the IPv4 destination in a packet capture. This minimal example checks the format and length, but does not validate the heartbeat CRC. For the observed V101 layout, compare stm32_crc(data[:108]) with the uint32 at offset 108 using Section 13.

If another program owns UDP 6789, close that listener or use its discovery output. Socket sharing behavior differs across operating systems.

7. C7FA Measurement Data Format

Point cloud data uses Ethernet UDP to the configured destination, factory host port 6668. The tested unit used source port 6544. Numeric fields are little-endian. Wire bytes c7 fa represent uint16 value 0xfac7. The earlier label "0xc7fa" described displayed bytes and was ambiguous as a numeric constant.

Offset	Field	Wire type	Meaning
0	code	uint16	Bytes c7 fa (LE value 0xfac7)
2	count	uint16	Number of points in this subpacket
4	whole_fan	uint16	Total points in the sector
6	offset	uint16	First point index within the sector
8	begin_ang	uint32	Sector start, 0.001 degree
12	end_ang	uint32	Sector end, 0.001 degree
16	flags	uint32	Status/configuration flags
20	timestamp	uint32	First-point time; hour-relative ms in driver
24	dev_no	uint32	Device number
28	distance[count]	uint16[]	All distances, mm
28+2*count	angle[count]	uint16[]	All point angle offsets, 0.001 degree
28+4*count	strength[count]	uint8[]	All intensity values
28+5*count	verify	uint16	Additive checksum

Expected datagram length = 30 + 5*count bytes. Arrays are consecutive: all distances, all angle offsets, then all intensities. They are not interleaved point records. Observed lengths included 490 bytes for 92 points, 70 bytes for 8 and 110 bytes for 16.

Point angle

```
angle_degrees[i] = (begin_ang + angle[i]) / 1000.0
```

The point angle is an offset from begin_ang, not a separate absolute bearing. Handle wrap-around when mapping angles into a 360-degree scan.

8. Scan Flags and Sector Geometry

Flag definitions depend on protocol/firmware version. Common low-bit definitions in the SDK are bit 0: mm, bit 1: intensity, bit 2: trailing-point removal, bit 3: smoothing; bit 8 describes rotation direction. Preserve raw flags when a bit interpretation is not confirmed.

Historical table - reference only

The original manual defines the following bits. In particular, its bits 4-7 conflict with SDK definitions below; this table is not a universal firmware contract.

Bit	Original manual definition for 1	For 0 / note
0	Millimeter units	Original default: mm
1	Intensity included	Original default: included
2	Trailing-point removal enabled	Disabled
3	Filtering enabled	Disabled
4	18-degree sector enabled	Disabled
5	9-degree sector enabled	Disabled
6	Other sector angle enabled	Disabled
7	Fixed angular resolution enabled	Disabled
8	Counterclockwise rotation	Clockwise
9-11	Reserved	Do not interpret

SDK differences and sector boundaries

The reviewed ROS2 driver's sdk/include/common_data.h names 0x10 DF_WITH_RESAMPLE, 0x20 DF_FAN_90, 0x40 DF_WITH_RPM and 0x80 DF_WITH_RESAMPLE_SOFT. Therefore, do not infer sector size from historical bits 4-7. Read begin_ang and end_ang. Observed flags 0x19f are not used here to infer undocumented bit meanings.

In the original RobiLidar-2D-01 example, ordinary sectors span 18 degrees and the first and last sectors span 36 degrees. The following sample covers 18 to 36 degrees. Treat this as example geometry, not a guarantee for all output modes. A packet with count < whole_fan is only part of a sector.

Before decoding arrays, validate prefix, minimum length, exact 30 + 5*count size and offset + count <= whole_fan. Apply reasonable limits before allocating memory. Reject truncated or inconsistent packets.

9. Original Scan Packet Example

The illustration includes bytes before the highlighted c7 fa payload, and the start of the next packet at its end. Only the C7FA payload participates in the scan checksum.

```
25 26 26 24 23 3E 04 C7 FA 5C 00 B7 00 00 00 50
46 00 00 A0 8C 00 00 03 01 00 00 FB C0 21 00 01
00 00 00 06 03 0A 03 0D 03 12 03 1A 03 1D 03 18
03 1F 03 1B 03 23 03 1F 03 2B 03 00 00 00 00 25
1B 28 1B 2C 1B 33 1B 33 1B 3C 1B 3D 1B 46 1B 4C
1B 50 1B 55 1B 5C 1B 5D 1B 62 1B 65 1B 67 1B 6E
1B 71 1B 76 1B 7C 1B 7F 1B 82 1B 87 1B 8C 1B 90
1B 94 1B 9C 1B 9D 1B 9F 1B A8 1B A8 1B B2 1B B8
1B BE 1B C0 1B C7 1B CC 1B D1 1B D3 1B D7 1B DA
1B DB 1B E2 1B E3 1B E7 1B EC 1B F4 1B FD 1B 01
1C 09 1C 14 1C 19 1C 1F 1C 22 1C 25 1C 39 1C 2D
1C 0F 1C 0E 1C 15 1C 1F 1C 22 1C 2E 1C 4C 1C 6A
1C 71 1C 7B 1C 7B 1C 81 1C 87 1C 89 1C 8E 1C 93
1C 98 1C A0 1C A8 1C B0 1C B9 1C 00 00 62 00 C4
00 26 01 88 01 EB 01 4D 02 AF 02 11 03 75 03 D7
03 39 04 97 04 F9 04 61 05 C3 05 26 06 88 06 EA
06 4C 07 AF 07 11 08 73 08 D5 08 38 09 9B 09 FD
09 60 0A C2 0A 24 0B 86 0B E9 0B 4B 0C AD 0C 0F
0D 71 0D D5 0D 37 0E 99 0E FA 0E 5E 0F C0 0F 22
10 85 10 E7 10 49 11 AB 11 0E 12 71 12 D2 12 34
13 97 13 FA 13 5C 14 BF 14 21 15 83 15 E5 15 47
16 AA 16 0C 17 6E 17 D2 17 34 18 96 18 F8 18 5B
19 BD 19 1F 1A 81 1A E4 1A 47 1B A9 1B 0A 1C 6E
1C D0 1C 32 1D 95 1D F7 1D 5A 1E BC 1E 1F 1F 80
1F E3 1F 45 20 A8 20 25 21 6C 21 CE 21 31 22 93
22 F5 22 21 1E 1E 1D 19 11 0B 0A 09 08 07 05 0D
0D 5A 58 59 58 57 56 55 54 55 56 64 66 66 65 64
64 63 64 66 65 64 65 65 63 63 62 63 62 61 5F 60
60 61 61 60 62 62 63 64 66 65 64 65 64 61 5F 5C
5B 5B 5B 5F 60 63 66 62 47 2C 2C 36 42 42 43 44
4C 5C 60 62 63 63 63 62 61 60 5D 5A 5A 5A 5A E9
9C C7 FA 5B 00 B7 00 5C 00 50 46 00 00 A0 8C 00
```

Packet Analysis

Bytes	Decoded field
c7 fa	Frame prefix
5c 00 / b7 00 / 00 00	count = 92 / whole_fan = 183 / offset = 0
50 46 00 00 / a0 8c 00 00	18000 = 18 degrees / 36000 = 36 degrees
03 01 00 00	flags = 0x00000103
fb c0 21 00 / 01 00 00 00	timestamp = 0x0021c0fb / device number = 1
06 03 ... / 00 00 ... / 21 ...	First distance 774 mm / first angle offset 0 / intensity 0x21
e9 9c	verify = 0x9ce9

This datagram contains 92 of 183 points in the sector. Additional subpacket data is required before the sector is complete.

10. Scan Checksum and Assembly

16-bit additive checksum

Sum modulo 65536 all 16-bit header words after the frame prefix, all distance words, all angle words, and each intensity byte zero-extended to 16 bits. Do not pair intensity bytes into checksum words. Compare with the trailing little-endian verify word. This formula was verified against captured scan packets; it is distinct from command CRC32.

```
import struct

def decode_c7(data):
    if len(data) < 30 or data[:2] != b"\xc7\xfa":
        raise ValueError("not a C7FA packet")
    count, whole, offset = struct.unpack_from("<HHH", data, 2)
    if len(data) != 30 + 5 * count or offset + count > whole:
        raise ValueError("invalid length or sector bounds")
    begin, end, flags, stamp, dev = struct.unpack_from("<IIIII", data, 8)
    distances = struct.unpack_from("<" + "H" * count, data, 28)
    angles = struct.unpack_from("<" + "H" * count, data, 28 + 2 * count)
    intensity = data[28 + 4 * count:28 + 5 * count]
    total = sum(struct.unpack_from("<13H", data, 2))
    total += sum(distances) + sum(angles) + sum(intensity)
    expected = struct.unpack_from("<H", data, 28 + 5 * count)[0]
    if (total & 0xffff) != expected:
        raise ValueError("scan checksum mismatch")
    degrees = [(begin + a) / 1000 for a in angles]
    return (whole, offset, begin, end, flags, stamp, dev,
            distances, degrees, intensity)
```

Original checksum example

```
0x9ce9 = (0x005c + 0x00b7 + 0x0000 + 0x0000 + 0x4650
 + 0x0000 + 0x8ca0 + 0x0000 + 0x0103 + 0x0021 + 0xc0fb
 + 0x0000 + 0x0001 + 0x0306 + ... + 0x1cb9 + 0x0000
 + ... + 0x22f5 + 0x0021 + ... + 0x005a) modulo 65536
```

Complete sector assembly

Associate fragments by source and matching sector headers: start/end angle, timestamp, device number and whole_fan. Place arrays at offset and track every point index. Reject inconsistent headers, conflicting overlaps and invalid bounds. Publish only with full coverage; discard incomplete assemblies after a timeout. If out-of-order support is required, implement and test it explicitly. Do not assume the SDK reorders arbitrary UDP arrivals.

11. LMSG Alarm Output Format

Alarm/status messages use ASCII LMSG on the scan/alarm upload channel. The tested unit produced 128-byte packets. Multibyte fields are little-endian. This preserves the original protocol version 0x101 description.

Offset	Field	Type	Meaning
0	sign	char[4]	Must be LMSG
4	proto_version	uint32	Protocol version, originally 0x101
8	dev_sn	char[20]	Device serial string
28	dev_id	uint32	Numeric device ID
32	timestamp	uint32	Timestamp
36	flags	uint32	Message type
40	events	uint32	Message content bit combination
44	id	uint16	Message number
46	extra	uint16	Extra payload length: 80
48	zone_actived	uint32	Currently active protection zone
52	all_states	uint8[32]	Equipment function status bytes
84	reserved	uint32[11]	Reserved; 44 bytes

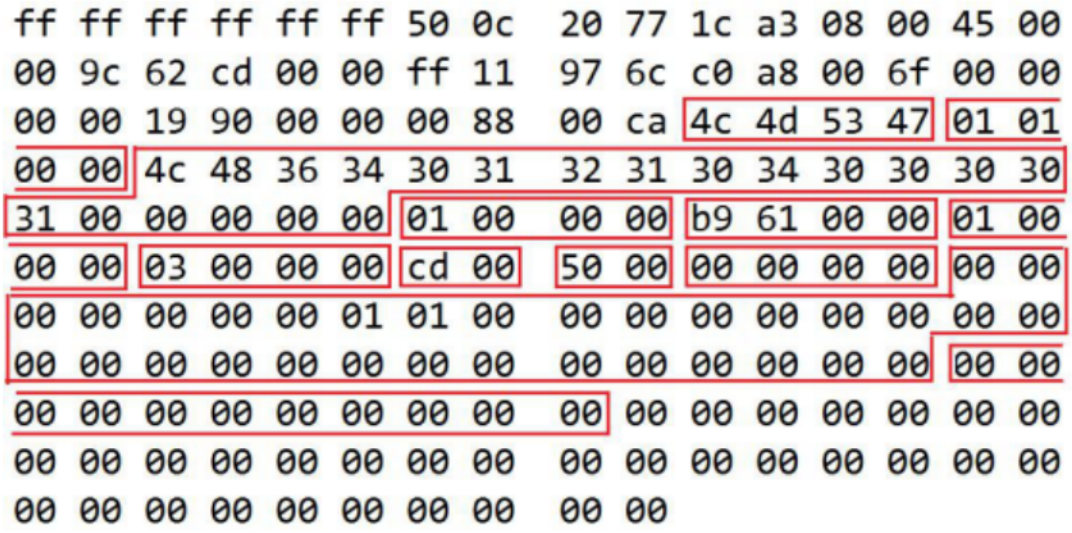
extra = 80 covers zone_actived (4 bytes) + all_states (32) + reserved (11*4). The complete message is 48 + 80 = 128 bytes. Check signature, version, declared length and available bytes before reading fields. This LMSG V101 layout defines no separate CRC field; reserved words are not a CRC. The older text reversed the serial-string and numeric-ID descriptions; the types above clarify their roles.

SDK definitions and model support

ROS2 driver's sdk/include/alarm.h defines message-type flags 1 error, 2 information, 4 debug and 0x100 alarm. Its error-event bits are 1 low power, 2 motor, 4 ranging-head overtemperature, 8 network, 0x10 ranging-head idle and 0x20 zero/missing data. These are SDK definitions, not a complete firmware contract. Preserve raw values when the unit-specific meaning is unknown.

Use alarm_msg:1 in the SDK configuration, or supported LSPST modes, to request alarm upload. Confirm actual LMSG traffic rather than treating send success as device acceptance.

12. Original Alarm Packet Example



Alarm packet parsing

Bytes	Interpretation
4c 4d 53 47 / 01 01 00 00	ASCII LMSG / protocol 0x00000101
4c 48 36 34 30 31 32 32 30 34 30 30 30 30 31 00 00 00 00 00	Serial string LH6401220400001
01 00 00 00	Numeric device ID 1
b9 61 00 00	Timestamp 0x000061b9, ms in original example
01 00 00 00	Message type 0x00000001
03 00 00 00	events = 3: insufficient power supply + motor stall in the original example
cd 00 / 50 00	Message number 0x00cd / extra length 80
00 00 00 00	Currently active zone 0
00 00 00 00 00 00 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Original 32 function-status bytes
00 00 00 00 00 00 00 00 00 00 00 ...	Reserved: 11 uint32 words, zero in the illustrated packet

The illustration includes lower-layer Ethernet/IP/UDP headers. Parse LMSG after those headers, or use a UDP socket that delivers the UDP payload directly. This historical sample is retained independently of the current device identity.

13. Command Frame and CRC32

Control/query traffic goes to device UDP port 6543. Binary framing wraps the ASCII command. Header fields and CRC trailer are little-endian.

Offset	Field	Meaning
0	sign: uint16	0x484c; wire bytes 4c 48
2	cmd: uint16	Packet type; verified queries use 0x0043
4	sn: uint16	Request sequence; match response
6	len: uint16	Payload length padded to a multiple of 4
8	payload[len]	ASCII command or binary data, zero-padded
8+len	crc: uint32	CRC32 over header + padded payload

STM32-style CRC32: polynomial 0x04c11db7, initial value 0xffffffff, no final XOR. Read each four-byte block as a little-endian uint32 word; XOR it into the CRC, then process 32 bits most-significant-bit first. Store the result as little-endian uint32. This differs from reflected Ethernet/zlib CRC32.

```
import struct

def stm32_crc(data):
    if len(data) % 4:
        raise ValueError("CRC requires complete 32-bit words")
    crc = 0xffffffff
    for (word,) in struct.iter_unpack("<I", data):
        crc ^= word
        for _ in range(32):
            if crc & 0x80000000:
                crc = ((crc << 1) ^ 0x04c11db7) & 0xffffffff
            else:
                crc = (crc << 1) & 0xffffffff
    return crc

def frame(packet_type, sequence, payload):
    payload = bytes(payload)
    padded_len = (len(payload) + 3) & ~3
    if padded_len > 0xffff:
        raise ValueError("payload too large")
    padded = payload + b"\0" * (padded_len - len(payload))
    body = struct.pack("<HHHH", 0x484c, packet_type,
                       sequence & 0xffff, padded_len) + padded
    return body + struct.pack("<I", stm32_crc(body))
```

Explicitly zero-fill your own padding. Do not transmit uninitialized memory or compute CRC over the unpadded ASCII command alone.

14. Verified Identity Queries

These read-only exchanges were verified on the unit identified in Section 1: request type 0x0043, response 0xffbc, matching sequence and valid CRC. Stop any driver/program occupying local UDP 6668 before running this example. Scans, LMSG and replies can share that socket; the next datagram is not necessarily an acknowledgment.

```
# Continue after stm32_crc() and frame() in Section 13.
import socket
import time

def parse_reply(data, expected_sn, expected_type=0xffbc):
    if len(data) < 12:
        raise ValueError("short reply")
    sign, kind, sn, length = struct.unpack_from("<HHHH", data)
    if (sign, kind, sn) != (0x484c, expected_type, expected_sn):
        raise ValueError("unrelated reply")
    if length % 4 or len(data) != 12 + length:
        raise ValueError("invalid reply length")
    got = struct.unpack_from("<I", data, 8 + length)[0]
    if stm32_crc(data[:8 + length]) != got:
        raise ValueError("reply CRC mismatch")
    return data[8:8 + length]

DEVICE = ("192.168.158.98", 6543)
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(("192.168.158.15", 6668))
for sequence, command in [(101, b"LUUIDH"), (102, b"LTYPEH")]:
    s.sendto(frame(0x0043, sequence, command), DEVICE)
    deadline = time.monotonic() + 2
    while True:
        remaining = deadline - time.monotonic()
        if remaining <= 0:
            raise TimeoutError("no matching reply")
        s.settimeout(remaining)
        data, source = s.recvfrom(65535)
        if source != DEVICE:
            continue
        try:
            payload = parse_reply(data, sequence)
        except ValueError:
            continue
        print(command, payload.rstrip(b"\0").decode("ascii", "replace"))
        break
```

Exact LUUIDH transaction

```
Request (sn=101): 4c484300650008004c5555494448000065c16300
Reply: 4c48bcff6500200050524f4455435420534e3a204c4831323632323630353030323231
0d0a0000003c0312ac
Payload: PRODUCT SN: LH1262260500221\r\n
LTYPEH reply payload: LDS-50C-E-R\0
```

For other commands, also verify the command-specific result or returned value. sendto() confirms local submission, not acceptance. Correlate source, sign, sequence, type, length, CRC and result.

15. Control Commands and Persistence

The SDK documents these generic commands. Confirm support and syntax for the supplied firmware. Only the identity query transactions above are marked verified here. Settings and start/stop commands affect the running device.

Command	Purpose / note
LSTARH / LSTOPH / LRESTH	Start / stop / restart operation
LUUIDH / LTYPEH	Query serial / model; verified on tested unit
LSRPM:<rpm>H	Set speed; official operating points 600/900 RPM (10/15 Hz). Confirm supported range with factory
LSUDP:%sH	Set IP, subnet mask, gateway, service port. Generic example: LSUDP:192.168.158.091 255.255.255.000 192.168.158.001 05000H
LSDST:%sH	Set upload destination IP and UDP port. Generic example: LSDST:192.168.158.015 06668H
LSSMT:%dH	Smoothing: 1 enable / 0 disable
LSDSW:%dH	Trailing-point removal: 1 enable / 0 disable
LSTFX:%dH	Fixed upload address: 1 enable / 0 disable
LSPST:%dH	Upload mode: 0 none / 1 data / 2 alarm / 3 data + alarm
LSDID:%dH	Set device ID; distinct from the SDK local handle

RPM packet type depends on the API path

Generic SDK documentation lists 450-1200 RPM, while SetRPM accepts 300-3000 RPM. Neither is a model guarantee. SetRPM sends C_PACK (0x0043); initialization and the ROS2 RPM setting use S_PACK (0x0053). Do not prescribe one universal packet type. Use the path supported by the target firmware and verify the result. SetRPM itself is not claimed as tested here.

Persistence and network changes

Official example/main.cpp comments describe RPM, smoothing and trailing-point removal as nonpersistent; network, destination, fixed upload, upload type and device ID as persistent. This is source-documented, not reboot-verified on the supplied unit. Record settings and confirm persistence after a controlled restart if needed.

According to the SDK, LSUDP may change the device IP without a reply. A timeout is not success. Test the new IP, inspect advertised settings and confirm packet destinations. An address change can interrupt the current socket session.

16. Host Keepalive and Timestamps

Host keepalive is separate from LiDA discovery

The host sends a framed request to device UDP 6543, type 0x4b41, with a 32-byte binary payload. The observed response was type 0xb4be with a 32-byte payload. Use Section 13 framing and CRC32. This is not a LiDA discovery packet.

```
struct KeepalivePayload {
    uint32_t world_clock;
    uint32_t mcu_hz;
    uint32_t arrive;
    uint32_t delay;
    uint32_t reserved[4];
};
```

world_clock is host milliseconds within the current hour. Other fields retain their SDK names; not all returned timing values have a verified interpretation. Zero-fill reserved words and validate the framed response before using fields.

```
# Requires frame() from Section 13.
import time
world_clock = int(time.time() * 1000) % 3_600_000
payload = struct.pack("<8I", world_clock, 0, 0, 0, 0, 0, 0, 0)
request = frame(0x4b41, 103, payload)
# Send to device_ip:6543. Correlate the 0xb4be response,
# source, sequence, length and CRC as for other command frames.
```

Scan time and rollover

The scan timestamp is hour-relative milliseconds in the driver, associated with the first point of the UDP packet. It is not an independent Unix timestamp. Absolute time reconstruction requires a synchronized hour reference and handling of the 3,600,000 ms rollover.

The driver supports host-time mode 0 and device-time mode 1. Host-time mode uses the host timing path; device-time mode requires appropriate synchronization and rollover handling. Do not simply append a date to a device value or interpret the two discovery time words as a verified epoch clock.

Operational checks

Monitor scans, discovery and command/keepalive replies independently. Even a response with valid CRC may contain a command-specific failure or unsupported result. After changing a setting, verify its returned value or actual scan/network behavior.

17. SDK and Troubleshooting

SDK: https://github.com/Robify-Robotics/RobiLidar-2D-01

Download SDK ZIP

Start with example/main.cpp: add sdk sources, include standard_interface.h, add the lidar from its configuration, register the callback and open the device. Use settings appropriate for the supplied unit.

Location	Reference
sdk/service/LidarCheckService.cpp	Discovery receiver and LiDA handling
sdk/Global.cpp	Framing, CRC and common transport
sdk/standard_interface.h / .cpp	Public interfaces and control API paths
example/main.cpp	Callback integration and persistence comments

Symptom	Checks
No scan data	Power/cabling; host IP; actual upload target; UDP 6668 bind/firewall. Same subnet does not fix a wrong target.
Discovery absent	UDP 6789 and correct interface; broadcast/firewall. Query the known IP and check scans separately.
IGMP but no LiDA	Membership is not sender traffic. Check the actual UDP destination; the tested unit broadcast LiDA.
Multiple adapters	Select connected Ethernet for routing/capture/group membership. Check overlapping subnets and routes.
Command timeout	Device 6543, local port conflicts, source/destination, type, sequence, length and CRC. Test changed IP before concluding success.
Incomplete sectors	Length and additive checksum; matching headers and offset coverage. Detect duplicates/missing fragments and expire incomplete assemblies.

Source/version note: original 11-page RobiLidar-2D-01 manual (mechanical, optical, specifications, wiring, scan/alarm/heartbeat examples); official SDK documentation and source, repository revision 8bfd6ad; ROS2 driver 2.6.4; observed traffic dated 27 October 2026. Historical samples are retained. Corrections clarify numeric frame code, frequency, temperature, timestamps and version-dependent flags. Generic command support and model mapping are not inferred beyond this evidence.