MindPalace Odin1 Lite User Manual

Document Version: V1.1.1 · Release Date: 2026-11-06

1. Product Overview

1.1 Product Introduction

MindPalace Odin1 Lite: Compact Spatial Memory Module

> **Designed for robots, drones, and next-generation intelligent devices**

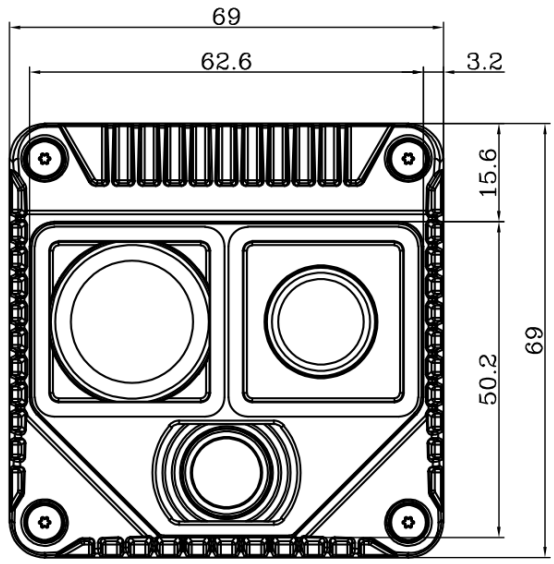
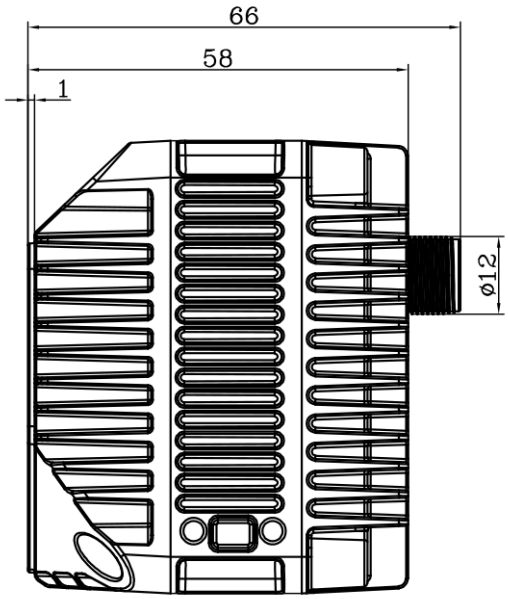
Deeply integrating LiDAR and vision sensors, it achieves high-precision, wide-angle real-time 3D perception. The built-in MindSLAM^{TM} high-performance fused SLAM algorithm delivers persistent spatial memory. It is ideal for rapid integration and deployment in size- and cost-sensitive robots, autonomous navigation platforms, and intelligent edge devices, operating reliably in complex indoor and outdoor environments.

1.2 Dimensions

Product Appearance

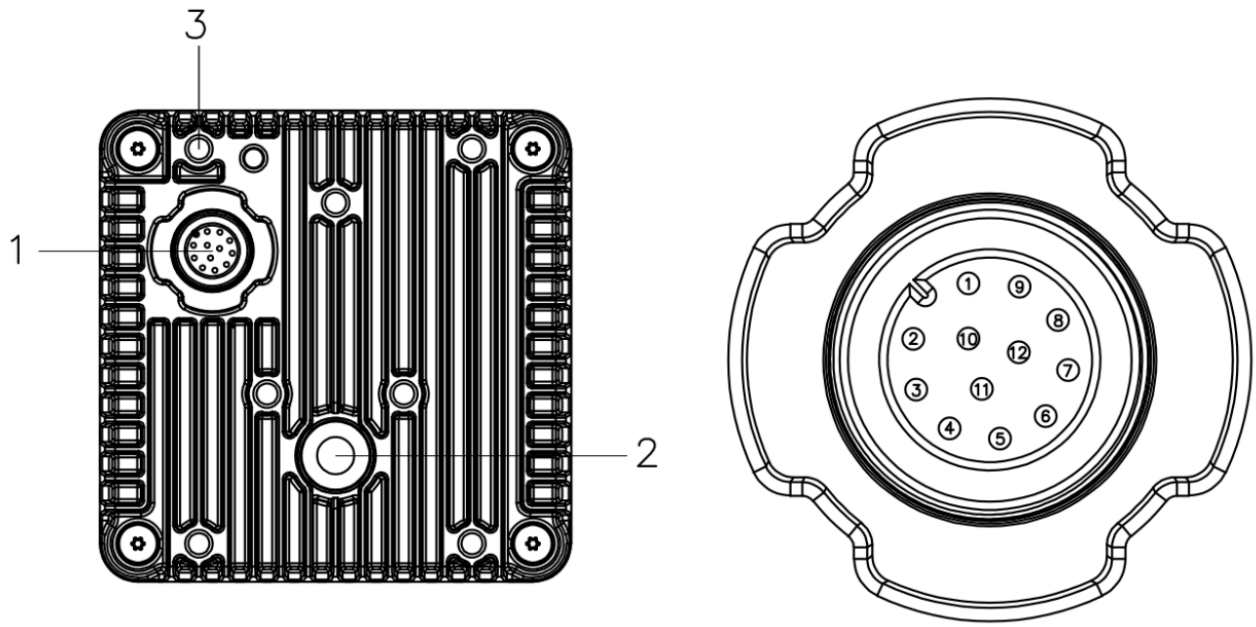


Product Dimensions

Front View	Right View
	

2. Hardware Description

2.1 Physical Interface



Odin1 Lite has a single circular connector. The included **Power & Data Y-Cable (circular connector)** provides both power and data transmission simultaneously:

No.	Interface	Cable	Description
1	Circular connector (device side)	Power & Data Y-Cable	Carries both power + Gigabit Ethernet data
2	DC plug (cable end)	5.5×2.1mm DC male plug	Connect to 12V/2A adapter
3	RJ45 (cable end)	Gigabit Ethernet	Connect to PC NIC or switch

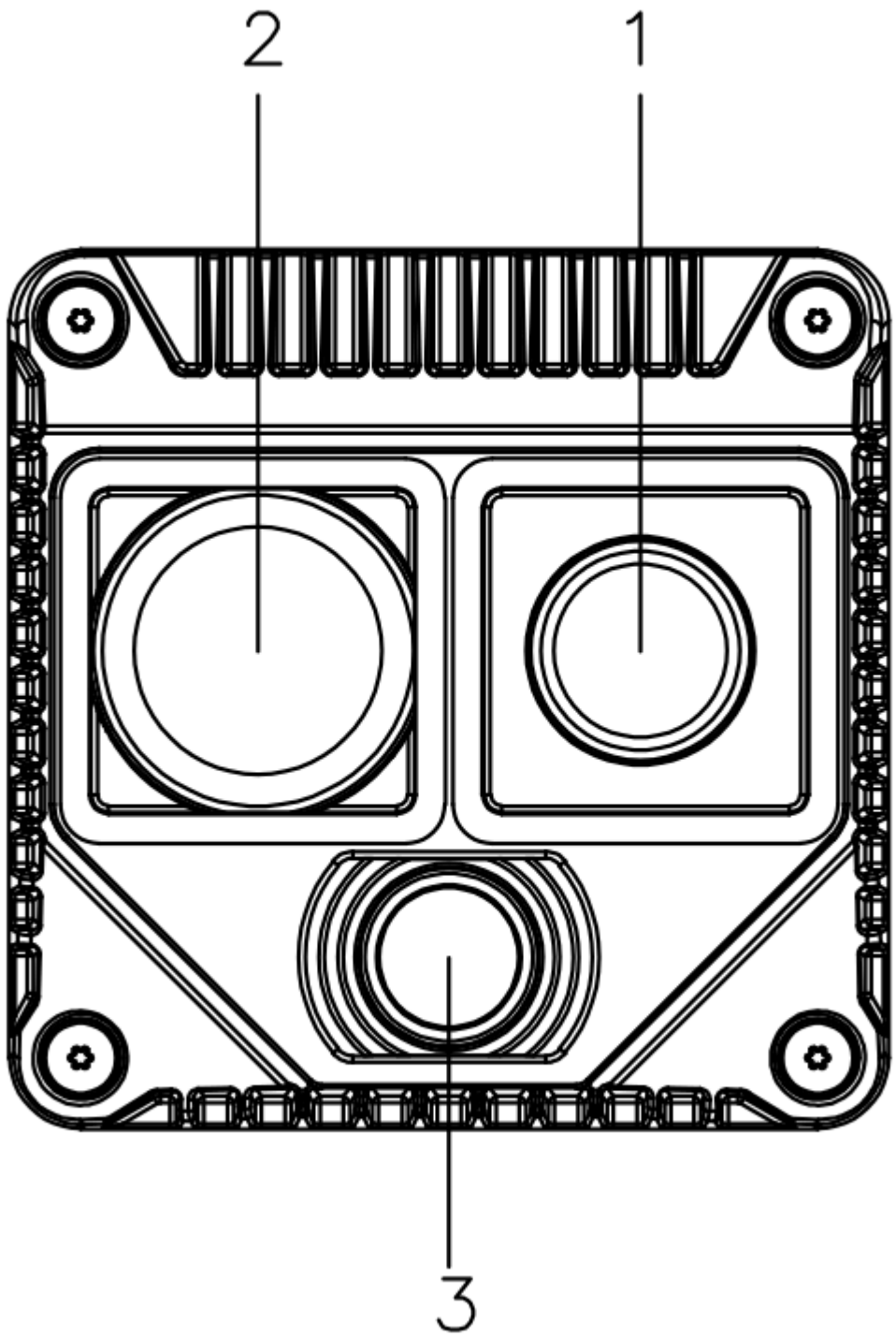
2.2 Pinout Definition

Odin1 Lite uses a **12-pin circular connector**, carrying power, Gigabit Ethernet, and external synchronization signals. The pinout is as follows:

Pin	Signal	Type	Function
1	DC+	Data	Gigabit Ethernet signal
2	VIN	Power	Power positive
3	DB−	Data	Gigabit Ethernet signal
4	DB+	Data	Gigabit Ethernet signal
5	DD+	Data	Gigabit Ethernet signal
6	DD−	Data	Gigabit Ethernet signal
7	DA+	Data	Gigabit Ethernet signal
8	DA−	Data	Gigabit Ethernet signal
9	DC−	Data	Gigabit Ethernet signal
10	GPIO_OUT	GPIO	External sync trigger signal input
11	GND	Power	Power negative
12	FSYNC_OUT	GPIO	Sync trigger signal output

> Gigabit Ethernet uses 4 differential pairs: DA±, DB±, DC±, DD±, conforming to the 1000BASE-T standard.

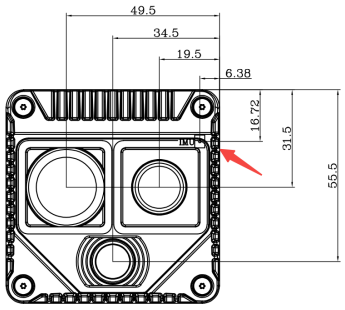
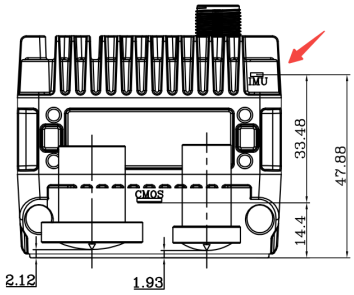
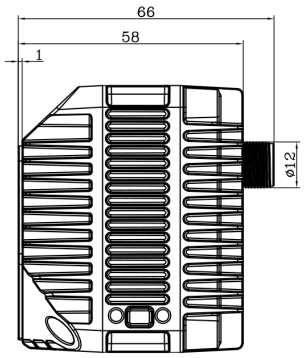
2.3 Core Components



No.	Component	Description
1	Laser Transmitter (TX)	dToF TX
2	Laser Receiver (RX)	dToF RX
3	RGB Camera	1280×1088

2.4 IMU Position Relationship

> **Note**: IMU and Body refer to the same physical location, only named differently. The following drawings show the IMU's installation position inside the device housing and its spatial relationship with other sensors.

Front View	Side View	Top View
		

Item	Description
IMU Location	The IMU is located inside the device, rigidly mounted on the main structure, sharing the same rigid body with the LiDAR optical center and RGB camera (CMOS)
Body Frame	The IMU coordinate frame is the Body coordinate frame; its origin is at the geometric center of the IMU chip
Housing Relationship	The IMU is rigidly fixed to the device's main structure; housing changes do not affect the extrinsics between IMU and LiDAR / camera
Mounting Orientation	The device supports mounting in any orientation; the IMU automatically calibrates to the gravity frame

2.5 Network Parameters

Item	Value	Notes
Factory IP	`192.168.1.251`	**Configurable**
Protocol	UDP (data) + HTTPS / mTLS (file transfer)	Ports 60000–60010, 59000–59005, 8080
HTTP Dashboard	`http://192.168.1.251:8080`	Device status / Calibration / OTA / Network config

2.6 CAD Models

> **TODO: Upload Odin1 Lite circular connector CAD files (stp) and update links**

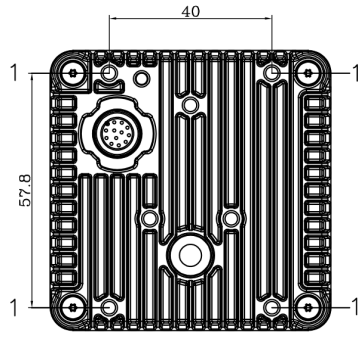
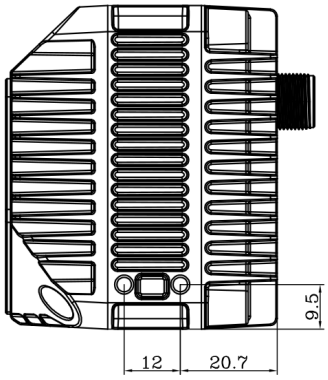
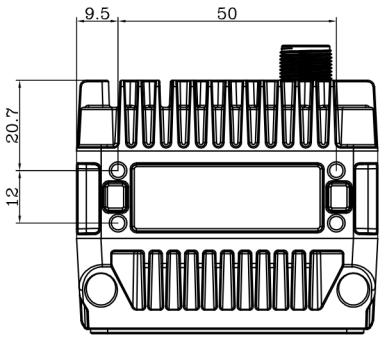
- Download Odin1 Lite connector CAD file (stp) — straight
- Download Odin1 Lite connector CAD file (stp) — right-angle

3. Installation Guide

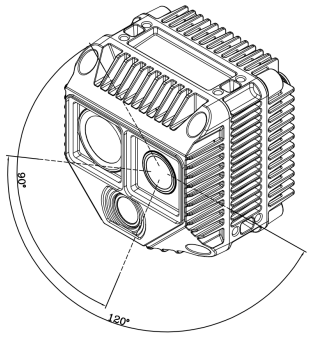
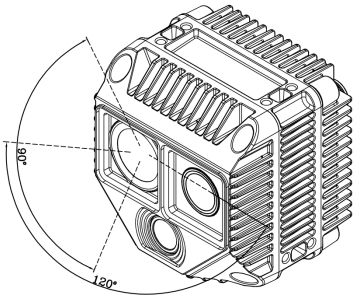
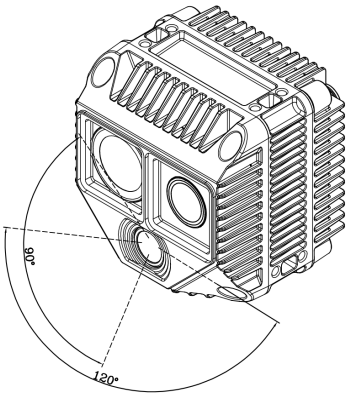
Before installing Odin1 Lite, please read the following precautions:

1. **Clean the installation surface and optical window:** - Before installation, thoroughly remove dust, oil, and other contaminants from the mounting area and the LiDAR optical window. Use a clean air blower to remove dust, then gently wipe the window with a lint-free cloth (e.g., lens cloth) slightly dampened with water. Residual contamination may affect measurement accuracy. Do not use harsh cleaning agents.
2. ⚠️ **Ensure unobstructed field of view:** - No objects may obstruct the sensor's front field of view (FOV). Note: even transparent materials such as glass or acrylic panels, if placed directly in front of the sensor, will significantly interfere with the laser signal and degrade performance.
3. **Flexible mounting orientation (with optimization tips):** - This product supports mounting in any orientation (including inverted). - The device automatically adapts to the installation posture (e.g., inverted, tilted); internal sensors calibrate to the gravity frame.
4. ⚠️ **Do not apply additional load:** - The device body (especially the heat dissipation housing and optical window) must not bear additional weight or pressure. Secure the device only through the designated mounting interfaces (e.g., bottom threaded holes).
5. ⚠️ **Ensure adequate cooling space:** - Sufficient clearance must be maintained around the device for heat dissipation. Requirement: minimum clearance ≥ 10mm between the device and surrounding objects to ensure unobstructed airflow and prevent overheating.
6. **Stable installation:** - It is strongly recommended to mount the device on a sturdy bracket, with the sensor bottom at least 0.2 m above any object to avoid field-of-view obstruction.
7. ⚠️ **Never reverse power polarity:** - Please carefully read the instructions and pinout definition.

3.1 Mechanical Mounting

Rear M3 Mounting Holes	Left/Right M3 Mounting Holes	Top/Bottom M3 Mounting Holes
		

Effective Field of View (FOV) range:

TX	RX	RGB
		

3.2 CAD Models

- Download Odin1 Lite CAD file (stp)
- Download Odin1 Lite CAD file with FOV (stp)

4. Power Supply & Connection

4.1 External Power Supply Design Guide

1. **Voltage Range Requirements**

The nominal operating voltage of Odin1 Lite is **DC 9-24V**. A **12V / 2A** DC adapter (≥ 24W power headroom) is recommended; otherwise, insufficient instantaneous current may prevent the device from starting or cause unstable operation.

2. **Power Adapter Notes**

- Connect the adapter via the DC plug of the **Power & Data Y-Cable**; the DC plug specification is **5.5×2.1mm**.
- Odin1 Lite **does not include a power adapter**; users must supply a compliant 12V adapter separately.

3. **Voltage Surge Protection Warning**

⚠️ Voltage fluctuations exceeding the rated upper limit on the power line may cause permanent damage to the device. It is recommended to add overvoltage protection circuitry in the power supply loop.

4. **Power Consumption Characteristics**

Operating State	Power
Steady-state operation	Typical ≈ **11** W
Cold-start peak	Maximum **24** W

5. **Key Power Supply Selection Parameters**

- Continuous output capability ≥ **2A**
- Recommended power headroom ≥ **24W**
- Overvoltage protection threshold recommended ≤ **26V**

4.2 Connection Steps

Odin1 Lite communicates with the PC via Ethernet.

1. Insert the circular connector of the **Power & Data Y-Cable** into the Odin1 Lite interface, aligning with the keyway, and tighten the threaded ring.
2. Connect the **DC plug** to the 12V/2A adapter and power on; a steady indicator light indicates successful power-up.
3. Connect the **RJ45 cable** to the PC network port (or switch).
4. Configure the PC NIC IP to `192.168.1.x/24` (same subnet as the device `192.168.1.251`).

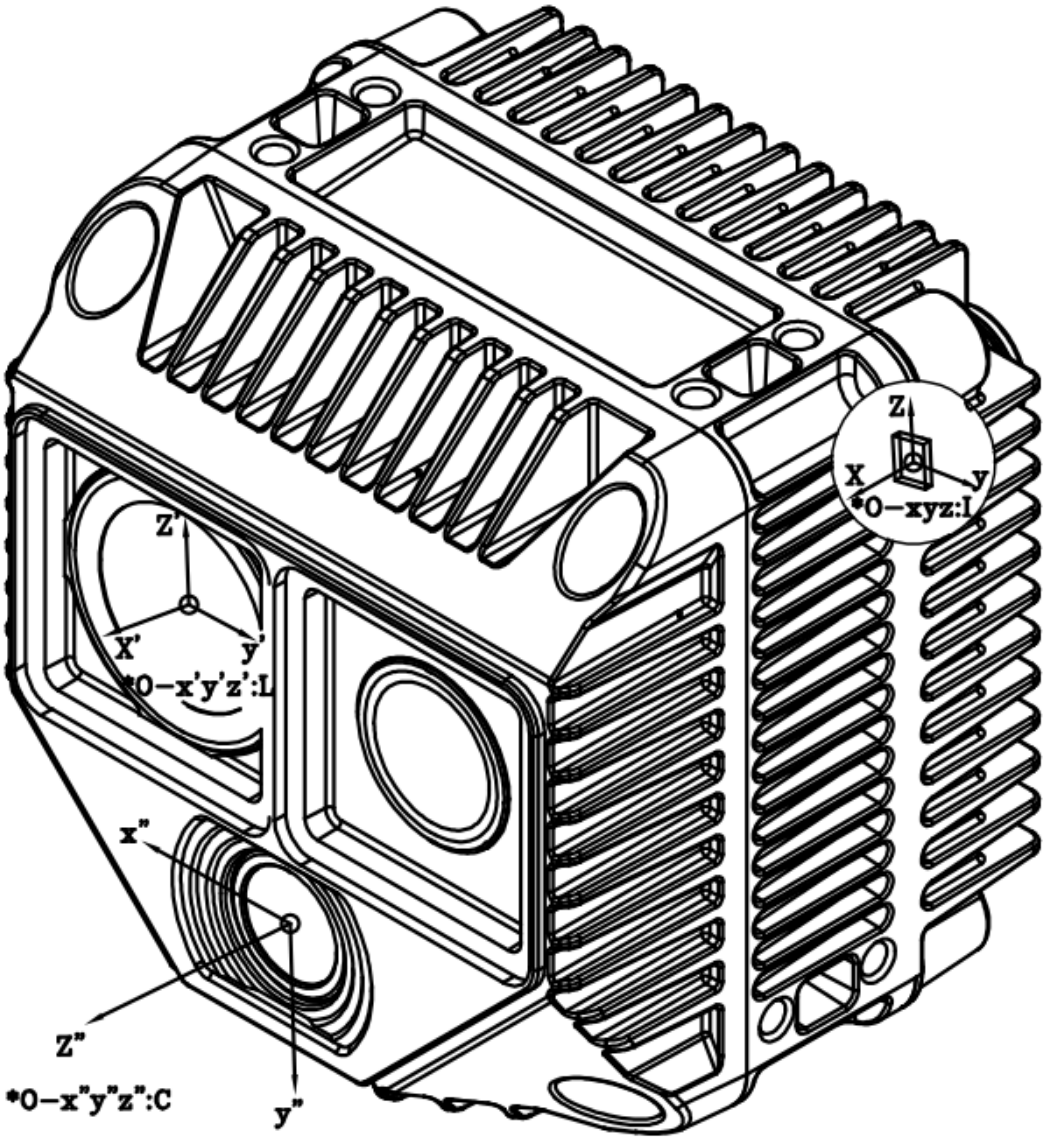
5. Verify connectivity with `ping 192.168.1.251`, then launch the driver or access the HTTP Dashboard.

> 🔧 For detailed network configuration steps, see 7. Quick Start §7.3.2 Network Configuration.

5. Data Output

5.1 Coordinate System Definition

The Cartesian coordinate system of Odin1 Lite is defined as shown below. **I** is the IMU (i.e., Body — both refer to the same physical location, only named differently) coordinate frame, **L** is the point cloud coordinate frame, and **C** is the camera coordinate frame.



5.2 Extrinsic Parameters

Extrinsic Parameter Notation

The notation $\mathbf{T}_{\mathbf{B}}^{\mathbf{A}}$ is used to represent coordinate transformations, specifically:

$$\mathbf{P_A} = \mathbf{T_B^A} \cdot \mathbf{P_B}$$

where P_A and P_B denote the coordinates of a point in frames A and B, respectively.

Camera - LiDAR

The extrinsic $\mathbf{T}^{\text{camera}}_{\text{lidar}}` varies between units. After launching the driver, retrieve it from config/calib.yaml (`Tcl_0`). See the driver documentation for details.

Body - LiDAR

> **Note**: **Body** here refers to the original **IMU** — only the naming (description) differs; they correspond to the same physical location.

The Body (formerly IMU) to LiDAR extrinsic $\mathbf{T}^{\text{body}}_{\text{lidar}}` (`body_T_lidar` in the calibration file) is as follows:

$$\mathbf{T}_{\text{lidar}}^{\text{body}} = \begin{bmatrix} 0.9999543679 & -0.0086157257 & 0.0041269153 & -0.0305398721 \\ 0.0086800624 & 0.9998369581 & -0.0158339418 & 0.0234679052 \\ -0.0039898215 & 0.0158690412 & 0.9998661185 & 0.0333162287 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Others

- For the Camera-to-Body extrinsic, please compute it from the above parameters.

5.3 Intrinsic Parameters

Intrinsic parameters primarily concern camera intrinsics; LiDAR intrinsics are not considered.

5.3.1 Odin1 Lite Camera Intrinsics

Methods to obtain intrinsics: - **Method 1 (Runtime)**: After the driver starts, it pulls the calibration file from the device and saves it to `~/catkin_ws/src/ros_driver/config/camera_calib_<SN>.yaml`, unique per device. - **Method 2 (ROS Service)**: Run `ros2 service call /manifold/device0/get_calibration odin_ros_driver_rev1/srv/GetCalibration "{}"` to retrieve the YAML string directly. - **Method 3 (HTTP Dashboard)**: Visit `http://192.168.1.251:8080` in a browser and download from Dashboard → Calibration File Management.

```
# O1L-P040100047
cam_num: 1
img_topic_0: /camera/rgb
Tcl_0: [
  0.01814, -0.99980, -0.00813, 0.00579,
  -0.00071, 0.00812, -0.99997, -0.03109,
  0.99984, 0.01815, -0.00057, -0.01629,
  0.00000, 0.00000, 0.00000, 1.00000
]

cam_0_640_544:
        cam_model: FishPoly
        image_width: 640
        image_height: 544
        k2: 3.9309765393149745e-02
        k3: 1.5817608030575755e-01
        k4: 5.8836968291029024e-01
        k5: -8.2827749950756613e-01
        k6: 3.8543134263557260e-01
        k7: -6.3564147672175983e-02
        p1: 0.
        p2: 0.
        A11: 2.3172737533779440e+02
        A12: -4.1424461677121709e-02
        A22: 2.3167070553813502e+02
        u0: 3.1331230979976016e+02
        v0: 2.7959256780073559e+02
        isFast: 0
        numDiff: 3000
        maxIncidentAngle: 120

cam_0_1280_1088:
        cam_model: FishPoly
        image_width: 1280
        image_height: 1088
        k2: 3.9309765393149745e-02
        k3: 1.5817608030575755e-01
        k4: 5.8836968291029024e-01
        k5: -8.2827749950756613e-01
        k6: 3.8543134263557260e-01
        k7: -6.3564147672175983e-02
        p1: 0.
        p2: 0.
```

```
A11: 4.6345475067558880e+02
A12: -8.2848923354243417e-02
A22: 4.6334141107627005e+02
u0: 6.2662461959952032e+02
v0: 5.5918513560147119e+02
isFast: 0
numDiff: 3000
maxIncidentAngle: 120
```

5.3.2 Camera Model Description

Odin1 Lite RGB Camera Model and Intrinsics

This section describes the projection model and intrinsic parameters of the Odin1 Lite RGB camera for engineering reference.

1 Camera Projection Model

Odin1 Lite RGB uses a **polynomial fisheye projection model** to describe wide-field imaging. After undistortion, this model can be simplified to a standard **pinhole model**.

2 Polynomial Fisheye Model

2.1 Projection Formula

Given a 3D point $\mathbf{P_c} = (X, Y, Z)^T$ in the camera coordinate frame, compute the incident radius:

$$r = \sqrt{X^2 + Y^2}$$

Compute the incident angle:

$$\theta = \arccos\left(\frac{Z}{\sqrt{X^2 + Y^2 + Z^2}}\right)$$

Polynomial radial mapping:

$$\theta_d = \theta + k_2\theta^2 + k_3\theta^3 + k_4\theta^4 + k_5\theta^5 + k_6\theta^6 + k_7\theta^7$$

Project onto the distortion plane:

$$x_d = \frac{\theta_d}{r} \cdot X, \quad y_d = \frac{\theta_d}{r} \cdot Y$$

2.2 Affine Transformation

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}$$

Parameter	Description
k2–k7	Polynomial distortion coefficients
A11	X-axis focal length (horizontal scaling)
A12	Non-orthogonal compensation (skew)
A22	Y-axis focal length (vertical scaling)
u0, v0	Principal point coordinates

2.3 Undistorted Pinhole Model

$$u = f_x \frac{X}{Z} + c_x, \quad v = f_y \frac{Y}{Z} + c_y$$

2.4 Coordinate Convention

Odin1 Lite RGB follows the OpenCV camera coordinate convention: X → right, Y → down, Z → forward (right-handed coordinate system).

2.5 Engineering Notes

- Undistortion does not change extrinsics.
- For high-precision reprojection, use the polynomial fisheye model.
- For detection, SLAM, and fusion algorithm frameworks, use the undistorted pinhole model.
- When $\theta \to 0$, numerical instability may occur ($r \to 0$ causes division by zero).

5.4 Data Description

Raw point cloud (`cloud/raw` , resolution 256×192) PointCloud2 fields:

Field	Data Type	Description
`x`	FLOAT32	X-axis, unit: meters
`y`	FLOAT32	Y-axis, unit: meters
`z`	FLOAT32	Z-axis, unit: meters
`intensity`	UINT8	Reflectance, range 0–255
`confidence`	UINT8	Point confidence, range 0–255. Higher = more reliable. Recommended threshold: 4 for indoor scenes; adjustable via `confidence_threshold` parameter
`offset_time`	FLOAT32	Time offset relative to the frame base timestamp, unit: seconds

> Note: When a point's `confidence` is below `confidence_threshold`, the driver sets x/y/z to 0. `point_step` = 18 bytes.

5.5 ROS Topics and Services Overview

> Package: `odin_ros_driver_rev1`, main node: `odin_ros_driver_node`, post-processing node: `post_process_node`.
>
> All topics and services are prefixed per device: `{prefix} = /{topic_prefix}/device{N}/` (default `/manifold/device0/`). Only suffixes are listed below.
>
> ℹ️ The device model never appears in the topic/service path. The model and SN are announced via the JSON fields of `/{topic_prefix}/driver/device_online`.
>
> ℹ️ `imu` appearing in topic/field names corresponds to **Body** in this document (same physical location, different naming only).

5.5.1 Main Driver Node (odin_ros_driver_node)

Topic Suffix	Type	Description
`cloud/raw`	`sensor_msgs/PointCloud2`	Raw point cloud (256×192)
`cloud/slam`	`sensor_msgs/PointCloud2`	SLAM point cloud
`camera0/compressed`	`sensor_msgs/CompressedImage`	Left RGB camera JPEG compressed image
`camera1/compressed`	`sensor_msgs/CompressedImage`	Right RGB camera JPEG compressed image (only on dual-camera devices)
`gray`	`sensor_msgs/Image`	Grayscale image (mono8)
`imu`	`sensor_msgs/Imu`	IMU (acceleration + angular velocity)
`odometry_lf`	`nav_msgs/Odometry`	Low-frequency odometry (with pose covariance)
`odometry_hf`	`nav_msgs/Odometry`	High-frequency odometry (with pose covariance)

The driver also publishes the following driver-level topics under `/{topic_prefix}/driver/` (shared across all devices):

Topic	Type	Description
`/{topic_prefix}/driver/device_online`	`std_msgs/String`	Device online event (latched, JSON with SN/model/prefix/IP)
`/{topic_prefix}/driver/device_offline`	`std_msgs/String`	Device offline event (latched, JSON with SN/model)
`/{topic_prefix}/driver/resolution_change`	`std_msgs/String`	Resolution-change event (latched)

5.5.2 Post-Processing Node (post_process_node)

Topic Suffix	Type	Description
`camera0/raw`	`sensor_msgs/Image`	Left RGB camera decoded image
`camera1/raw`	`sensor_msgs/Image`	Right RGB camera decoded image (only on dual-camera devices)
`camera0/undistort`	`sensor_msgs/Image`	Left RGB camera undistorted image
`camera1/undistort`	`sensor_msgs/Image`	Right RGB camera undistorted image (only on dual-camera devices)
`camera0/rect`	`sensor_msgs/Image`	Stereo-rectified left image
`camera1/rect`	`sensor_msgs/Image`	Stereo-rectified right image (only on dual-camera devices)
`camera0/camera_info`	`sensor_msgs/CameraInfo`	Rectified left camera intrinsics
`camera1/camera_info`	`sensor_msgs/CameraInfo`	Rectified right camera intrinsics
`cloud/render`	`sensor_msgs/PointCloud2`	Colored point cloud (point cloud + image fusion, LiDAR frame)
`reprojection/overlay`	`sensor_msgs/Image`	Point cloud reprojected onto camera image overlay
`pointcloud2depth/depth`	`sensor_msgs/Image` (32FC1)	Dense depth map, units in meters; `0` = invalid pixel
`pointcloud2depth/depth_color`	`sensor_msgs/Image` (bgr8)	JET-colored depth map
`pointcloud2depth/depth_overlay`	`sensor_msgs/Image` (bgr8)	Depth map overlaid on camera image

> ℹ️ Odin1 Lite is monocular RGB. Topics such as `camera1/*`, `stereo/*`, etc. reserved for stereo/disparity point clouds in the driver will not publish data on Odin1 Lite.

5.5.3 Services

Service Suffix	Type	Description
`get_calibration`	`odin_ros_driver_rev1/srv/GetCalibration`	Retrieve camera calibration YAML content
`set_sensor_mode`	`odin_ros_driver_rev1/srv/SetSensorMode`	Set Sensor Mode (0=HDR / 1=High Peak / 2=Low Peak / 3=Adaptive)
`reset_initial_pose`	`odin_ros_driver_rev1/srv/ResetInitialPose`	Reset SLAM initial pose, map, and factor graph
`get_adaptive_param` / `set_adaptive_param`	`GetAdaptiveParam` / `SetAdaptiveParam`	Read / set Adaptive-mode parameters
`get_sync_mode` / `set_sync_mode`	`GetSyncMode` / `SetSyncMode`	Read / set the default time synchronization mode
`get_ptp_sync_status`	`odin_ros_driver_rev1/srv/GetPtpSyncStatus`	Query PTP synchronization status
`restore_default_config`	`odin_ros_driver_rev1/srv/RestoreDefaultConfig`	Restore device factory configuration; restart required
`device_mode`	`odin_ros_driver_rev1/srv/DeviceMode`	Set device mode (0=Standby / 1=Normal / 2=Upgrade / 3=SLAM / 4=Relocation / 5=ReMap)
`save_map`	`odin_ros_driver_rev1/srv/SaveMap`	Save loop-closure map to a host file
`load_map`	`odin_ros_driver_rev1/srv/LoadMap`	Load a map from a host file for relocalization or ReMap
`exploy_map`	`odin_ros_driver_rev1/srv/ExployMap`	Export the current loop-

Service Suffix	Type	Description
		closure map to a host-side file (format depends on plugin)
`set_diag_monitor`	`odin_ros_driver_rev1/srv/SetDiagMonitor`	Enable / disable 1 Hz frame-rate and latency diagnostics
`set_channel_config` / `get_channel_config`	`SetChannelConfig` / `GetChannelConfig`	Configure / query main-driver data channels
`set_post_process_config` / `get_post_process_config`	`SetPostProcessConfig` / `GetPostProcessConfig`	Configure / query post-processing channels

5.5.4 Point Cloud Transport Compression

`cloud/raw` and `cloud/slam` are published through `point_cloud_transport`. The base topics are always plain `sensor_msgs/PointCloud2` with **RELIABLE + VOLATILE** QoS and can be subscribed directly in RViz / `ros2 topic echo` without any extra setup.

After installing compression plugins on the driver machine and restarting the driver, the following compressed sub-topics are also published:

Topic	Type	Description
`{prefix}cloud/raw/zlib`	`point_cloud_interfaces/CompressedPointCloud2`	zlib lossless compression
`{prefix}cloud/raw/zstd`	`point_cloud_interfaces/CompressedPointCloud2`	zstd lossless compression
`{prefix}cloud/raw/draco`	`point_cloud_interfaces/CompressedPointCloud2`	Google Draco lossy compression

Install the plugins (ROS2 example; replace `humble` with your distro):

```
sudo apt install ros-humble-point-cloud-transport-plugins
```

> RViz cannot decode compressed point clouds directly. To visualize them, run a `republish` node on the subscriber machine or use a `point_cloud_transport` subscriber in your own code. See the driver `docs/topics.md` for details.

5.6 ROS Driver Download

Odin1 Lite ROS Driver: https://git.manifoldtech.cn/manifoldtech-sdk/odin_ros_driver2

For detailed deployment and runtime instructions, see 7. Odin1 Lite Quick Start (Ubuntu).

6. Mapping and Relocalization API

6.1 Overview

This document describes how to use ROS Services for Odin1 Lite mapping (SLAM) and relocalization, along with typical workflows.

> Relocalization requires both a ROS Driver matching the device firmware and the `manifold-lc-sdk` algorithm deb package. The ROS Driver alone provides sensor data but cannot run mapping or relocalization.

6.2 Concepts

Term	Description
Device	Odin1 Lite module
Host	PC / robot connected to Odin1 Lite and running the ROS driver
Device Mode	Device operating mode, switched via the `device_mode` service
Normal (1)	Normal odometry mode; device outputs odom / point cloud, etc.
SLAM (3)	Mapping mode; the algorithm builds a loop closure map while running
Relocation (4)	Relocalization mode; loads a saved map for position relocalization
ReMap (5)	Relocalization + incremental mapping mode; extends a loaded map while localizing
Map File	Binary map file (`.bin`) generated by the algorithm in SLAM / ReMap mode

6.3 Related ROS Services

> All services are prefixed with `/{topic_prefix}/device{N}/` (default `/manifold/device0/`).
>
> ℹ️ The device model never appears in the topic/service path. The model and SN are announced via the JSON fields of `/{topic_prefix}/driver/device_online` .

Service	Type	Description
`device_mode`	`DeviceMode`	Switch device mode: 0=Standby / 1=Normal / 2=Upgrade / 3=SLAM / 4=Relocation / 5=ReMap
`save_map`	`SaveMap`	Save current loop-closure map to a specified path
`load_map`	`LoadMap`	Load saved map file for relocalization or ReMap
`exploy_map`	`ExployMap`	Export the current loop-closure map to a host-side file (format depends on plugin)

Service Definitions

DeviceMode.srv

```
uint8 mode   # 0=Standby 1=Normal 2=Upgrade 3=SLAM 4=Relocation 5=ReMap
---
bool   success
int32  error_code   # 0=OK, otherwise SDK OdinResult enum
string message
```

SaveMap.srv

```
string path   # Output map file path (host local path)
---
bool   success
int32  error_code
string message
```

LoadMap.srv

```
string path   # Input map file path (host local path)
---
bool   success
int32  error_code
string message
```

6.4 Typical Workflows

6.4.1 Mapping Workflow

Step 1: Launch the driver

```
# ROS2
ros2 launch odin_ros_driver_rev1 odin1Lite.launch.py

# ROS1
roslaunch odin_ros_driver_rev1 odin1Lite.launch
```

Step 2: Switch to SLAM mode (start mapping)

> Before executing, enter the ROS workspace and source the environment: ```bash cd ~/catkin_ws source install/setup.bash # ROS2
> # source devel/setup.bash # ROS1
> ```

```
# ROS2
ros2 service call /manifold/device0/device_mode \
  odin_ros_driver_rev1/srv/DeviceMode "{mode: 3}"

# ROS1
rosservice call /manifold/device0/device_mode "mode: 3"
```

The device now enters SLAM mapping mode. Move the device to begin mapping.

Step 3: Save the map

After mapping is complete, call `save_map` to save the map to a host-local file:

```
# ROS2
ros2 service call /manifold/device0/save_map \
  odin_ros_driver_rev1/srv/SaveMap "{path: '/tmp/odin_map.bin'}"

# ROS1
rosservice call /manifold/device0/save_map "path: '/tmp/odin_map.bin'"
```

> Saving the map does not stop mapping; you can continue and save again.

Step 4: Stop mapping (optional)

Switch back to Normal mode:

```
# ROS2
ros2 service call /manifold/device0/device_mode \
  odin_ros_driver_rev1/srv/DeviceMode "{mode: 1}"

# ROS1
rosservice call /manifold/device0/device_mode "mode: 1"
```

6.4.2 Relocalization Workflow

Prerequisite: A map file has been saved via the mapping workflow above.

Step 1: Switch to Relocation mode

> Before executing, enter the ROS workspace and source the environment: ```bash cd ~/catkin_ws source install/setup.bash # ROS2
> # source devel/setup.bash # ROS1
> ```

```
# ROS2
ros2 service call /manifold/device0/device_mode \
  odin_ros_driver_rev1/srv/DeviceMode "{mode: 4}"

# ROS1
rosservice call /manifold/device0/device_mode "mode: 4"
```

Step 2: Load the map

```
# ROS2
ros2 service call /manifold/device0/load_map \
  odin_ros_driver_rev1/srv/LoadMap "{path: '/tmp/odin_map.bin'}"

# ROS1
rosservice call /manifold/device0/load_map "path: '/tmp/odin_map.bin'"
```

After successful loading, the device begins running the relocalization algorithm. In RViz, set **Fixed Frame** to `device0/map` so the loaded map is displayed. Move the device and confirm convergence from the application-side localization state and pose output. Loop-closure visualization topics (see §6.4.4) are also published during optimization.

Step 3: Stop relocalization

```
# ROS2
ros2 service call /manifold/device0/device_mode \
  odin_ros_driver_rev1/srv/DeviceMode "{mode: 1}"

# ROS1
rosservice call /manifold/device0/device_mode "mode: 1"
```

6.4.3 ReMap Workflow (Incremental Mapping)

ReMap (`mode: 5`) localizes against a previously saved map and **extends** it at the same time. Use this when you want to add new areas to an existing map.

Step 1: Switch to ReMap mode and load the existing map

```
# ROS2
ros2 service call /manifold/device0/device_mode \
  odin_ros_driver_rev1/srv/DeviceMode "{mode: 5}"

ros2 service call /manifold/device0/load_map \
  odin_ros_driver_rev1/srv/LoadMap "{path: '/tmp/odin_map.bin'}"
```

Set RViz **Fixed Frame** to `device0/map` to see the loaded map.

Step 2: Move the device to collect new areas

The algorithm will localize against the old map and incrementally build new map content.

Step 3: Save the extended map

```
ros2 service call /manifold/device0/save_map \
  odin_ros_driver_rev1/srv/SaveMap "{path: '/tmp/odin_map_extended.bin'}"
```

> The saved ReMap file is usually larger than the original map because it contains the incremental extensions.

Step 4: Stop ReMap

```
ros2 service call /manifold/device0/device_mode \
  odin_ros_driver_rev1/srv/DeviceMode "{mode: 1}"
```

6.4.4 Loop-Closure Visualization Topics

When using SLAM / Relocalization / ReMap, the driver publishes loop-closure visualization topics under the device prefix:

Topic Suffix	Type	Description
`loop_closure/optimization_busy`	`std_msgs/Bool`	Whether the optimizer is currently running
`loop_closure/optimization_status_marker`	`visualization_msgs/Marker`	Text marker showing `IDLE` / `OPTIMIZATION`
`loop_closure/optimized_path`	`nav_msgs/Path`	Optimized trajectory in the `local_map` frame
`loop_closure/optimized_cloud`	`sensor_msgs/PointCloud2`	Optimized global point cloud in the `local_map` frame
`loop_closure/map_cloud`	`sensor_msgs/PointCloud2`	Loaded map point cloud in the `map` frame

> These topics are produced by the `manifold-lc-sdk` algorithm package. They will only appear when the algorithm deb package is installed and the device is in SLAM / Relocalization / ReMap mode.

6.5 Workflow Summary

Workflow	`device_mode`	Key Services	Result
SLAM mapping	`3`	`device_mode` → `save_map` → `device_mode` (1)	New map file
Relocalization	`4`	`device_mode` → `load_map` → `device_mode` (1)	Localized pose against existing map
ReMap (incremental)	`5`	`device_mode` → `load_map` → `save_map` → `device_mode` (1)	Extended map file

6.6 Notes

1. The `path` parameter for `save_map` and `load_map` is a **host local path** — ensure the path is writable / readable.
2. An empty path returns `error_code=2` (`kInvalidArgument`); a disconnected device or an unready algorithm session returns `error_code=1` (`kNotInitialized`).
3. `save_map` must be called in SLAM (`mode: 3`) or ReMap (`mode: 5`) mode; `load_map` must be called in Relocation (`mode: 4`) or ReMap (`mode: 5`) mode.

4. When `mode` is outside the `0–5` range, `kInvalidArgument` is returned.
5. Typical workflows: - Mapping: `device_mode mode: 3` → `save_map` → `device_mode mode: 1` - Relocalization: `device_mode mode: 4` → `load_map` → `device_mode mode: 1` - ReMap: `device_mode mode: 5` → `load_map` → `save_map` → `device_mode mode: 1`
6. A successful service call updates the driver's desired mode for recovery after reconnecting.

7. Odin1 Lite Quick Start (Ubuntu)

> This chapter guides users through running the Odin1 Lite ROS driver on Ubuntu to obtain point cloud, image, IMU, odometry, and other data.

7.1 Firmware / Driver Version Compatibility

No.	Firmware Version	Driver Version
1	v1.3.7	v0.7.0
2	v2.0.x	v0.12.0
3	v3.x	v3.x (latest: v3.7.0)

> Ensure firmware and driver versions match; version mismatches may cause functional issues. `v3.x` firmware must use a `v3.x` driver, not v0.12.0. The following service, parameter, and topic descriptions are based on v3.7.0.

7.2 Hardware Components & Installation

#	Component	Description
1	Odin1 Lite unit	Integrated LiDAR + monocular RGB + IMU module
2	Power & Data Y-Cable	One end connects to the unit via circular connector; the other end splits into DC plug and RJ45
3	12V/2A DC power adapter	User-supplied (≥ 24W recommended)
4	Ethernet cable (direct or via switch)	Connects Odin1 Lite to PC

Installation Steps:

1. Align the circular connector keyway with the Odin1 Lite interface, insert, and tighten the threaded ring.
2. On the other end of the Y-cable: connect the DC plug to the 12V/2A adapter, and the RJ45 connector to the PC network port (or switch).
3. After power-up, the device indicator light should be steadily on, indicating normal operating status.

> ⚠️ Confirm keyway alignment before tightening the circular connector to avoid bending the pins.

7.3 PC Setup

7.3.1 System Requirements

Ubuntu Version	Supported ROS Version
20.04	ROS1 Noetic
22.04	ROS2 Humble
24.04	ROS2 Jazzy

> Ensure the correct ROS version is installed and `source` d before running the driver.

7.3.2 Network Configuration

Odin1 Lite communicates with the PC via **Ethernet**. The default IP is `192.168.1.251` (module IP can be customized per customer requirements). Configure the PC NIC to the same subnet (e.g., `192.168.1.100/24`).

```
# 1. Verify NIC IP is active
ifconfig

# 2. Test device connectivity
ping 192.168.1.251
```

> If ping fails, re-enable the NIC or check IP / subnet mask / firewall for UDP broadcast blocking (ports 60000-60010).

7.3.3 Network Buffer & ROS_DOMAIN_ID

Before running the driver, increase the kernel network-buffer limits to avoid packet drops caused by high-rate sensor data, and set a unique `ROS_DOMAIN_ID` when multiple PCs share the same LAN.

```
# Increase receive buffer sizes (required for stable streaming / mapping)
sudo sysctl -w net.core.rmem_max=16777216
sudo sysctl -w net.core.rmem_default=8388608
sudo sysctl -w net.core.netdev_max_backlog=50000

# Set an isolated ROS2 domain (1-99; 0 means shared)
export ROS_DOMAIN_ID=1
```

> The `export ROS_DOMAIN_ID=...` command only affects the current terminal. Add it to `~/.bashrc` to make it permanent. For ROS1, see 7.14 Multi-Host Isolation.

7.4 Obtaining the Driver

Steps:

1. Contact Manifold after-sales technical support
2. Provide company information, user information, and email
3. Technical support will provide a GitLab account
4. Log in to GitLab with the provided account to download the driver (change the password promptly after first login)

Driver repository: https://git.manifoldtech.cn/manifoldtech-sdk/odin_ros_driver2

> The latest driver is `v3.7.0`, matching `v3.x` firmware. For v1.3.7 or v2.0.x firmware, use the compatible historical driver release.

Driver archive naming convention:

```
ros_driver_v<version>_<release_type>_<date>.tar.xz
```

Example: `ros_driver_v3.7.0_release_20261024.tar.xz`

7.5 Deploying the Driver

The following example assumes the driver package is downloaded to `~/Downloads` with a workspace named `catkin_ws`.

```
# 1. Create ROS workspace
mkdir -p ~/catkin_ws/src

# 2. Extract driver and move to src
cp ~/Downloads/ros_driver_v3.7.0_release_20261024.tar.xz ~/catkin_ws/src
cd ~/catkin_ws/src
tar -xvf ros_driver_v3.7.0_release_20261024.tar.xz
mv ./ros_driver_v3.7.0_release_20261024/ros_driver ./
rm -rf ros_driver_v3.7.0_release_20261024 ros_driver_v3.7.0_release_20261024.tar.xz
```

The resulting directory structure should be:

```
catkin_ws/
└── src/
    └── ros_driver/
        ├── script/build_ros1.sh
        ├── script/build_ros2.sh
        ├── launch/
        ├── config/
        └── ...
```

7.6 Building the Driver

> The SDK (`module/sdk_api`) is automatically built by the build scripts — no separate installation is needed.

ROS1 (Noetic)

```
source /opt/ros/noetic/setup.bash
cd ~/catkin_ws
./src/ros_driver/script/build_ros1.sh
```

ROS2 (Humble / Jazzy)

```
source /opt/ros/humble/setup.bash       # or jazzy
cd ~/catkin_ws
./src/ros_driver/script/build_ros2.sh
```

7.7 Running the Driver

ROS1

```
cd ~/catkin_ws
source devel/setup.bash

# Default launch (auto-discover device + RViz)
roslaunch odin_ros_driver_rev1 odin1Lite.launch

# Specify image resolution and frame rate
roslaunch odin_ros_driver_rev1 odin1Lite.launch \
    image_width:=640 image_height:=544 image_fps:=30

# Launch without RViz
roslaunch odin_ros_driver_rev1 odin1Lite.launch start_rviz:=false
```

ROS2

```
cd ~/catkin_ws
source install/setup.bash

# Default launch
ros2 launch odin_ros_driver_rev1 odin1Lite.launch.py

# Specify image resolution and frame rate
ros2 launch odin_ros_driver_rev1 odin1Lite.launch.py \
    image_width:=640 image_height:=544 image_fps:=30

# Launch without RViz
ros2 launch odin_ros_driver_rev1 odin1Lite.launch.py start_rviz:=false
```

After successful launch, point cloud and images will be displayed in RViz automatically.

Launch Parameters

Parameter	Default	Description
`start_rviz`	`true`	Whether to launch RViz
`topic_prefix`	`manifold`	Root topic prefix; resulting prefix is `/{topic_prefix}/device{N}/`
`device_mode`	`normal`	Startup device mode: `normal` or `standby`
`discovery_timeout_ms`	`2000`	SDK device-discovery timeout in ms
`image_width` / `image_height` / `image_fps`	`0`	Image dimensions and frame rate; `0` keeps the device capability
`image_format`	`mjpeg`	Image format: `mjpeg` or `h264`
`filter_confidence_threshold`	`-1`	Device-side confidence filter; `-1` keeps the device value
`filter_adaptive_cdf_percentile`	`-1`	Adaptive filter CDF percentile (`1..10000` , 0.01%); `-1` keeps device value
`filter_adaptive_outdoor_thr`	`-1`	Adaptive filter outdoor distance threshold in mm; `-1` keeps device value
`filter_range_min_depth`	`-1`	Range-filter minimum valid depth in mm; `-1` keeps device value
`raw_frame_id` / `gray_image_frame_id`	`lidar`	`frame_id` for raw cloud and grayscale image
`slam_frame_id` / `odom_frame_id`	`odom`	`frame_id` for SLAM cloud and odometry parent frame
`image_frame_id` / `image2_frame_id`	`camera0` / `camera1`	Camera image frame IDs
`imu_frame_id`	`imu`	IMU frame ID
`enable_row_filter`	`false`	Enable row filter (resolution 256×192, rows `0..191`)
`min_row`	`0`	Row-filter minimum row; points with row < `min_row` are zeroed
`tf_publish_rate`	`0`	Periodic TF republish rate in Hz; `0` disables it
`confidence_threshold`	`7`	Initial point-cloud confidence threshold (`0..255`)
`sensor_mode`	`-1`	Initial dToF sensor mode; `-1` keeps the current device mode
`log_level`	`1`	Initial SDK / ROS log level (`0` =ERROR, `1` =INFO, `2` =WARN, `3` =DEBUG)

7.8 Parameter Reference

Parameters are either **startup parameters**, read only when a node starts, or **runtime parameters**, which take effect immediately. Except for the parameters below and the data-stream toggles listed in 7.11 Data Stream Switches, setting other values with `ros2 param set` only changes the parameter server; restart the driver with launch arguments to apply them.

Runtime parameters (hot-switchable)

Parameter	Range / Default	Description
`confidence_threshold`	`0..255` / `7`	Point-cloud confidence threshold; start with `4` indoors and raise only as needed
`sensor_mode`	`-1..3` / `-1`	dToF mode; `-1` keeps the current device mode at startup
`log_level`	`0..3` / `1`	SDK / ROS log level: `0`=ERROR, `1`=INFO, `2`=WARN, `3`=DEBUG

The following A-layer channel switches on the main driver node and B-layer switches on the post-processing node are also hot-switchable (see §7.11):

- Main driver: `enable_raw_point`, `enable_slam_point`, `enable_image0`, `enable_image1`, `enable_imu`, `enable_odom_lf`, `enable_odom_hf`, `enable_tf`
- Post-processing: `send_image_raw`, `send_image2_raw`, `send_cloud_render`, `send_undistort`, `send_rectify`, `send_opencv_undistort`

```
# ROS2: inspect and modify runtime parameters
ros2 param list /odin_ros_driver_node
ros2 param get /odin_ros_driver_node sensor_mode
ros2 param set /odin_ros_driver_node sensor_mode 3
ros2 param set /odin_ros_driver_node confidence_threshold 15

# ROS2: hot-switch a data channel
ros2 param set /odin_ros_driver_node enable_image0 false
ros2 param set /postprocess send_undistort true

# ROS1: use rqt_reconfigure or dynamic_reconfigure
rosrun rqt_reconfigure rqt_reconfigure
```

> The actual main-driver node name may include the device SN (e.g., `/odin_driver_P000000001`). Use `ros2 node list | grep odin_driver_` to find it.

7.9 Sensor Mode (Long/Short Frame) Switching

Odin1 Lite supports four operating modes:

Enum	Mode	Description
0	HDR	High dynamic range, auto-balances near and far
1	High Peak (long frame)	Optimizes reception under strong light, **best for outdoor**, slightly lower close-range accuracy
2	Low Peak (short frame)	Optimizes close-range and small-space noise, **best for indoor**, affected by strong light
3	Adaptive	Adaptive mode

Method 1: rqt GUI

```
rqt
# In the Dynamic Reconfigure panel, locate odin_ros_driver_node
# and adjust the sensor_mode parameter.
```

Method 2: Runtime parameter (ROS2)

```
source ~/catkin_ws/install/setup.bash

# Switch to High Peak
ros2 param set /odin_ros_driver_node sensor_mode 1

# Switch to Low Peak
ros2 param set /odin_ros_driver_node sensor_mode 2
```

7.10 Mapping and Relocalization

> For detailed API documentation, see 6. Mapping and Relocalization API.
>
> Relocalization requires both the ROS Driver matching the device firmware and the `manifold-lc-sdk` algorithm deb package. The ROS Driver alone can publish sensor data but cannot run mapping or relocalization.
>
> Run `download_lc_sdk.sh` on the host to identify the suitable deb package, download the `RECOMMENDED` package shown by the script, then install it with `sudo dpkg -i manifold-lc-sdk_<version>_<architecture>.deb` .
>
> Ensure firmware is updated to the latest version.

7.10.1 Mapping (SLAM)

```
# 1. Ensure the driver is running and the device is connected

# 2. Switch to SLAM mode (start mapping)
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 3}"

# 3. Move the device to map...

# 4. Save the map
ros2 service call /manifold/device0/save_map \
    odin_ros_driver_rev1/srv/SaveMap "{path: '/tmp/odin_map.bin'}"

# 5. Mapping complete, switch back to Normal
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 1}"
```

7.10.2 Relocalization

```
# 1. Switch to Relocation mode
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 4}"

# 2. Load the previously saved map
ros2 service call /manifold/device0/load_map \
    odin_ros_driver_rev1/srv/LoadMap "{path: '/tmp/odin_map.bin'}"
```

Set RViz **Fixed Frame** to `device0/map` to display the loaded map. Move the device and confirm convergence from the application-side localization state and pose output.

```
# 3. Stop relocalization, switch back to Normal
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 1}"
```

7.10.3 ReMap (Incremental Mapping)

ReMap (`mode: 5`) localizes against an existing map while extending it. Use this to add new areas to a previously saved map.

```
# 1. Switch to ReMap mode and load the existing map
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 5}"

ros2 service call /manifold/device0/load_map \
    odin_ros_driver_rev1/srv/LoadMap "{path: '/tmp/odin_map.bin'}"

# 2. Move the device to collect new areas

# 3. Save the extended map
ros2 service call /manifold/device0/save_map \
    odin_ros_driver_rev1/srv/SaveMap "{path: '/tmp/odin_map_extended.bin'}"

# 4. Stop ReMap
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 1}"
```

> The saved ReMap file is usually larger than the original map because it contains the incremental extensions.

7.10.4 Loop-Closure Visualization Topics

When the algorithm deb package is installed and the device is in SLAM / Relocation / ReMap mode, the following topics are published:

Topic Suffix	Type	Description
`loop_closure/optimization_busy`	`std_msgs/Bool`	Whether the optimizer is running
`loop_closure/optimization_status_marker`	`visualization_msgs/Marker`	`IDLE` / `OPTIMIZATION` text marker
`loop_closure/optimized_path`	`nav_msgs/Path`	Optimized trajectory in `local_map` frame
`loop_closure/optimized_cloud`	`sensor_msgs/PointCloud2`	Optimized global point cloud in `local_map` frame
`loop_closure/map_cloud`	`sensor_msgs/PointCloud2`	Loaded map point cloud in `map` frame

7.10.5 Confirming Relocalization Results

A successful `load_map` response only confirms that the map has been loaded into the algorithm session. Move the device a short distance and confirm convergence from the application-side localization state and pose output.

7.11 Data Stream Switches

All data-stream switches have been moved from `control_command.yaml` to launch parameters. The A-layer and B-layer switches below are hot-switchable at runtime via `ros2 param set` / `rqt_reconfigure` or the `set_channel_config` / `set_post_process_config` services. The C/D/E-layer switches are startup-only.

A-layer: main-driver channel switches (`odin_ros_driver_node`)

Parameter	Default	Description
`enable_raw_point`	`true`	Raw point cloud channel
`enable_slam_point`	`true`	SLAM point cloud channel
`enable_image0`	`true`	Left RGB camera image channel
`enable_image1`	`false`	Right RGB camera image channel (ignored on Odin1 Lite)
`enable_imu`	`true`	IMU data channel
`enable_odom_lf`	`true`	Low-frequency odometry channel
`enable_odom_hf`	`true`	High-frequency odometry channel
`enable_tf`	`true`	TF transform channel

B-layer: post-processing switches (`post_process_node`)

Parameter	Default	Description
`post_process`	`true`	Master switch for the post-processing node (startup only)
`send_image_raw`	`true`	Publish decoded left RGB image
`send_image2_raw`	`false`	Publish decoded right RGB image (ignored on Odin1 Lite)
`send_cloud_render`	`false`	Publish colored point cloud
`send_undistort`	`false`	Publish undistorted RGB images
`send_rectify`	`false`	Publish stereo-rectified images + `camera_info`
`send_opencv_undistort`	`false`	Enable the OpenCV `cv::fisheye` undistortion node

C/D/E-layer: startup-only feature switches

Parameter	Node	Default	Description
`enable_reprojection`	`pointcloud_reprojector_node`	`false`	Enable sparse point-cloud reprojection overlay
`enable_pointcloud_to_depth`	`pointcloud_to_depth_node`	`false`	Enable point-cloud → dense depth map
`enable_depth_to_pointcloud`	`depth_to_pointcloud_node`	`false`	Enable depth map → dense colored point cloud

```
# Hot-switch a single channel
ros2 param set /odin_ros_driver_node enable_raw_point false
ros2 param set /postprocess send_undistort true

# Batch-switch main-driver channels via service
ros2 service call /manifold/device0/set_channel_config \
    odin_ros_driver_rev1/srv/SetChannelConfig \
    "{enable_raw_point: -1, enable_slam_point: 0, enable_image0: -1, enable_image1: -1,
enable_imu: -1, enable_odom_lf: -1, enable_odom_hf: -1, enable_tf: -1}"

# Batch-switch post-processing outputs via service
ros2 service call /manifold/postprocess/set_post_process_config \
    odin_ros_driver_rev1/srv/SetPostProcessConfig \
    "{send_image_raw: -1, send_image2_raw: -1, send_cloud_render: -1, send_undistort: 1,
send_rectify: -1, send_opencv_undistort: -1}"
```

> Service fields use `0` =off, `1` =on, `-1` =keep current. Unlisted fields default to `0`, so explicitly set every field you do **not** want to change to `-1`.
>
> ℹ️ Odin1 Lite is monocular RGB. `enable_image1`, `send_image2_raw`, and stereo/depth-to-pointcloud related parameters will not produce data on Odin1 Lite.

Recording ROS2 rosbags

Before recording, make sure `ROS_DOMAIN_ID` is configured and the kernel receive buffers are enlarged (see 7.3.3 Network Buffer & ROS_DOMAIN_ID). For ROS2, use the QoS override file shipped with the driver to avoid dropping high-frequency topics:

```
ros2 bag record -a \
  --qos-profile-overrides-path ~/catkin_ws/src/ros_driver/config/rosbag2_qos.yaml
```

ROS1 uses TCPROS and does not require a QoS override file; `rosbag record -a` is sufficient.

7.12 Odin HTTP Dashboard

The device has a built-in web console — no additional software required.

- URL: `http://192.168.1.251:8080` (ensure PC and device are on the same subnet)
- Recommended browsers: Chrome / Edge

Feature	Page	Description
Device Info	Dashboard	Product name, firmware version, serial number
System Resource Monitor	Dashboard	CPU, memory, system load (1/5/15 min)
Temperature Charts	Dashboard	SoC and sensor temperature (1Hz line chart)
Frame Rate Charts	Dashboard	Real-time FPS for each sensor
Calibration File Management	Dashboard	Calibration file upload / download
Map File Download	Dashboard	Relocalization map file download
System Log Download	Dashboard	Device log archive (.tar.gz)
Telemetry Recording	Dashboard	Real-time monitoring data export to CSV
OTA Firmware Upgrade	Update	Drag-and-drop firmware → auto-install → reboot verification
Network Configuration	Setting	IP / gateway / subnet mask / DHCP
Device Reboot	Setting	Remote device reboot
Theme Switch	Global	Light / dark mode
Admin Login	Global	Admin privileges for advanced operations

7.13 Firmware Upgrade

1. Contact Manifold technical support to obtain the firmware package
2. Extract the firmware package and ensure the `.bin` file is visible
3. Upgrade steps: - Ensure PC and Odin1 Lite are on the same subnet and can `ping` each other - Open `http://192.168.1.251:8080` in a browser - Click **Update** in the sidebar - Click **Browse Files** and select the extracted `.bin` file - Wait for the upgrade to complete - Click **Dashboard** in the sidebar to verify the firmware version

7.14 Multi-Host Isolation (Avoiding Topic Conflicts)

When multiple PCs on the same LAN run the driver simultaneously, identical topics will interfere with each other.

ROS2: Set independent `ROS_DOMAIN_ID`

```
echo $ROS_DOMAIN_ID           # Check; empty or 0 = shared
export ROS_DOMAIN_ID=1        # 1–99; 0 = shared
```

ROS1: Restrict to localhost or use a different port

```
# Recommended
export ROS_LOCALHOST_ONLY=1

# Or: use a different port
roscore -p 11312
export ROS_MASTER_URI=http://localhost:11312
```

> The above commands apply only to the current terminal. For permanent effect, add them to `~/.bashrc` .

8. Firmware Upgrade

Odin1 Lite supports two upgrade methods: **HTTP Dashboard (recommended)** and **SDK CLI tool**.

8.1 Firmware Acquisition

Download the latest Odin1 Lite firmware package and release notes from the page below.

Item	Info
Firmware Download	https://version.manifoldtech.cn/download/o1l
File Format	`.bin` (HTTPS OTA / SDK CLI compatible)

8.2 Firmware Release Notes

Please consult Manifold technical support for firmware version release notes.

8.3 Companion ROS Driver Release Notes (v3.7.0 / 20261016)

1. Added loop-closure visualization topics: publishes loaded map, keyframe correction result, optimized result, and odometry trajectory.
2. Added ReMap mode (`device_mode=5`, `OdinDeviceMode` adds `kReMap`): supports extending a loaded map while localizing; host-side loop closure switches to ReMap; firmware data source handling is consistent with SLAM; `SaveMap` / `LoadMap` updated to support ReMap; added corrected loop-closure point cloud publishing.
3. Restored `map_cloud` callback registration; RViz uses a three-level color scale to distinguish loaded, corrected, and optimized point clouds, and shows continuous odometry trajectory.
4. Fixed `map_cloud` coordinate frame and RViz trajectory retention time; restored separation of SDK algorithm control and callback mutex locks.
5. ROS service return messages and logs now use `OdinResultStr` enum names instead of raw error codes.
6. Added `ExployMap` loop-closure map export interface: loop-closure plugin ABI, SDK `ExployMap` API, and ROS1/ROS2 `exploy_map` service passthrough; specific export format depends on plugin implementation.
7. Added OpenSSL and crypto static libraries.

Notable changes since v3.2.0

- **v3.6.0 / 2026-09-30**:

- Added FFmpeg H.264 decoder runtime hardware acceleration auto-detection and dynamic decoder discovery (NVDEC/CUVID/VAAPI/VDPAU), with automatic fallback to software decoding on failure.
- Added VAAPI/VDPAU hardware decode support (Intel/AMD GPU); device nodes can be specified via `ODIN_CUDA_DEVICE` / `ODIN_VA_DEVICE` / `ODIN_VDPAU_DEVICE`.
- Added optional libyuv integration to accelerate YUV→BGR conversion (~7× faster than libswscale/OpenCV for H.264/JPEG decode and NV12 composition paths); disable at runtime with `ODIN_DISABLE_LIBYUV=1`.
- Added device-specific launch files: `odin1Lite.launch(.py)` (mono RGB) and `odin2Special.launch(.py)` (dual RGB).
- **BREAKING**: Topic/service paths no longer include the device model. Per-device prefix changed from `/{topic_prefix}/{model}/device{N}/` to `/{topic_prefix}/device{N}/` (e.g., `/manifold/ODIN2/device0/` → `/manifold/device0/`).
- `device_online` JSON now includes a `model` field; `device_offline` changed from a plain SN string to JSON.
- **v3.2.0 / 2026-08-22**:
- Removed `control_command.yaml`; stream toggles moved to launch parameters.
- Added `set_channel_config` / `get_channel_config` and `set_post_process_config` / `get_post_process_config` services for batch hot-switching.
- Added `opencv_undistort_node`.
- Added row filter (`enable_row_filter`, `min_row`) for `cloud/raw`.
- Fixed `point_cloud_transport` plugin publishing.
- **v3.1.0 / 2026-08-13**:
- **BREAKING**: Launch parameter `operating_mode` renamed to `device_mode`.
- **BREAKING**: `image_format` options adjusted: removed `yuyv` / `nv21` / `rgb24`, added `h264`; now supports `mjpeg|h264`.
- **BREAKING**: `sensor_mode` default changed from `1` (High Peak) to `-1` (keep device current mode).
- **BREAKING**: Sensor `frame_id` defaults changed to semantic names (`lidar`, `camera0`, `camera1`, `imu`, etc.).
- Added ROS services: `set_sync_mode` / `get_sync_mode`, `set_adaptive_param` / `get_adaptive_param`, `reset_initial_pose`, `restore_default_config`, `get_ptp_sync_status`, `set_diag_monitor`.
- Added device-side filter launch parameters and `tf_publish_rate`.
- **v3.0.x and earlier**: Initial v3.x firmware support, dense depth / colored point cloud nodes, reprojection, MJPEG hardware decode on Jetson/Rockchip, ROS2 Jazzy support.

8.4 Method 1: HTTP Dashboard (Recommended)

> ⚠️ Do not run the ROS driver during upgrade. Do not disconnect power. A stable wired network connection is recommended.

1. Visit `http://192.168.1.251:8080` in a browser (ensure PC and device are on the same subnet).
2. Open the **Update** page and **drag the firmware** `.bin` into the upload area (or click to select).
3. The system automatically verifies the signature and installs; a progress bar is displayed during the process.
4. After installation, the device **auto-reboots**; once the Dashboard is accessible again, verify on the Dashboard page: - **Firmware version** has been updated to the target version - **Device serial number** is consistent - **Temperature / frame rate** are within normal range

8.5 Method 2: SDK CLI Tool

Suitable for server / automated deployment scenarios. Prerequisite: SDK is built (`module/sdk_api/build/odin_upgrade`).

```
# Auto-discover device and upgrade
./build/odin_upgrade /path/to/firmware_v<version>.bin

# Specify device IP
./build/odin_upgrade /path/to/firmware.bin --device 192.168.1.251

# Specify discovery timeout
./build/odin_upgrade /path/to/firmware.bin --timeout 5000
```

The upgrade link uses **mTLS mutual authentication + Ed25519 signature**, fully encrypted; signature verification failure immediately aborts installation without damaging the device.

8.6 Calibration File Management (Also via HTTPS)

- HTTP Dashboard → **Dashboard → Calibration File Management**: Upload / download `camera_calib_<SN>.yaml`
- CLI tool:

`bash # Retrieve calibration ./build/odin_get_calibration --device 192.168.1.251 --output ./camera_calib.yaml # Flash calibration ./build/odin_send_calibration ./camera_calib.yaml --device 192.168.1.251`

9. Relocalization User Guide

9.1 Overview

Odin1 Lite supports the following device modes, switched through the `device_mode` ROS service:

mode	Mode	Description
0	Standby	Standby
1	Normal	Normal operation (default)
2	Upgrade	Upgrade mode
3	SLAM	Mapping mode
4	Relocation	Relocalization mode
5	ReMap	Relocalization + incremental mapping

> Relocalization / ReMap requires both a ROS Driver matching the device firmware and the `manifold-lc-sdk` algorithm deb package. The ROS Driver alone can publish sensor data but cannot run mapping, relocalization, or incremental mapping.

9.2 Usage Tutorial

9.2.1 Prerequisites

1. Build and launch the ROS Driver matching the device firmware.
2. Run `download_lc_sdk.sh` on the host, download the `RECOMMENDED` `manifold-lc-sdk` deb link reported by the script, and install it:

`bash chmod +x download_lc_sdk.sh ./download_lc_sdk.sh sudo dpkg -i manifold-lc-sdk_<version>_<architecture>.deb`

The script reports suitable download links only; it does not download or install the package.

3. Save a usable map file in SLAM mode and note its host-local path.
4. Source the ROS environment:

`bash cd ~/catkin_ws source install/setup.bash`

9.2.2 Mapping (SLAM)

```
# Step 1: Switch to SLAM mapping mode
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 3}"

# Step 2: Move the device to collect map data...

# Step 3: Save the map after collection is complete
ros2 service call /manifold/device0/save_map \
    odin_ros_driver_rev1/srv/SaveMap "{path: '/tmp/odin_map.bin'}"

# Step 4: Switch back to Normal mode
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 1}"
```

> ❗ Do not exit the driver with `Ctrl+C` during mapping — you must call `save_map` first to save the map.

9.2.3 Relocalization

```
# Step 1: Switch to Relocation mode
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 4}"

# Step 2: Load the previously saved map
ros2 service call /manifold/device0/load_map \
    odin_ros_driver_rev1/srv/LoadMap "{path: '/tmp/odin_map.bin'}"

# Step 3: Move the device and confirm convergence from the application-side localization
state and pose output

# Step 4: After relocalization, switch back to Normal
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 1}"
```

9.2.4 Confirming Relocalization Results

A successful `load_map` response only confirms that the map has been loaded. Move the device a short distance and confirm convergence from the application-side localization state and pose output.

> ⚠️ Odin1 Lite cannot perform relocalization while stationary. It is recommended to move the device a short distance after loading the map.

9.3 Notes

- Map saving requires the algorithm loop closure session to be ready (device connected and calibration available); when the session is not ready, the service returns a non-zero `error_code`
- When `mode` is outside the 0–5 range, the service returns `kInvalidArgument`
- When the device is not connected, the service returns `kNotInitialized`
- Use `ros2 service list` to confirm actual service names; in v3.x the device model no longer appears in the topic/service path

10. Relocalization Map Acquisition Guide

10.1 Overview

Odin1 Lite provides relocalization capability and supports the following methods for acquiring relocalization maps (`.bin` files):

Method	Description
1	Use Odin1 Lite SLAM mode to directly generate bin files (recommended)
2	Use a Manifold handheld scanner to collect lx data, then export bin files via Mindcloud

10.2 Method 1: Direct Mapping via SLAM Mode (Recommended)

This is the simplest method and can be completed via ROS services:

```
cd ~/catkin_ws
source install/setup.bash

# 1. Switch to SLAM mapping mode
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 3}"

# 2. Move Odin1 Lite to collect environment data...

# 3. Save the map after collection is complete
ros2 service call /manifold/device0/save_map \
    odin_ros_driver_rev1/srv/SaveMap "{path: '/tmp/odin_map.bin'}"

# 4. Switch back to Normal
ros2 service call /manifold/device0/device_mode \
    odin_ros_driver_rev1/srv/DeviceMode "{mode: 1}"
```

The map file will be saved to the specified path (e.g., `/tmp/odin_map.bin`).

> ❗ Do not exit the driver with `Ctrl+C` during mapping — you must call `save_map` first.

10.3 Method 2: Manifold Handheld Scanner lx Collection + Mindcloud Export

If you do not have a Manifold handheld device, please choose another method.

1. Connect Q9000 to your phone and use Mindcloud Go for data scanning
2. After scanning, export the project file to local storage
3. Use Mindcloud to process the lx file; in post-processing, click the "Export Map" button to obtain the relocalization bin file

10.4 Map Download (HTTP Dashboard)

Relocalization maps can also be downloaded via the built-in HTTP Dashboard:

1. Visit `http://192.168.1.251:8080` in a browser
2. On the Dashboard page, find the "Map File Download" feature
3. Download the bin file to local storage

11. Maintenance and Troubleshooting

11.1 Routine Maintenance

1. **Cleaning Guidelines:** - Use an air blower to remove lens dust - Use a clean soft cloth to wipe the device housing - ⚠️ Do not use corrosive solvents to clean the window glass. Corrosive solvents will damage the glass surface coating; if corrosive solvent enters through the window glass edge gap, it will corrode the glass waterproof adhesive, causing waterproofing failure. Prohibited corrosive solvents include but are not limited to: alcohol, isopropanol, naphtha, turpentine, etc.
2. **Storage Conditions:** - Temperature: -20°C ~ 60°C - Humidity: < 85% RH

11.2 Common Troubleshooting

> For more issues, see 13. FAQ.

Symptom	Possible Cause	Solution
No response after power-up	Power supply issue	Check 12V adapter output, DC plug connection, circular connector lock
`ping 192.168.1.251` fails	PC and device not on same subnet / cable / NIC issue	Set PC NIC to `192.168.1.x/24`; restart NIC; replace cable
HTTP Dashboard inaccessible	Port 8080 blocked by firewall / device not fully booted	Disable firewall or allow port 8080; wait ~30s and retry
No data output after driver launch	UDP broadcast blocked by firewall / incorrect subnet	Allow UDP 60000–60010 and confirm the PC is on the same subnet. Device discovery, NIC selection, and ports are handled by the SDK; do not set `device_ip`.
RViz reports TF transform failure	Source TF frequency is too low for interpolation	Start with `tf_publish_rate:=50`; 50–200 Hz is recommended.
`post_process_node` crashes on its first image	System OpenCV and ROS `cv_bridge` link to incompatible versions	Rebuild `vision_opencv` / `cv_bridge` against the system OpenCV, then rebuild the driver.
Relocalization always failing	Device stationary / map path invalid / algorithm session unavailable	Move the device, verify the map path, and confirm `manifold-lc-sdk` is installed.
High packet loss / mapping drift	NIC receive buffer too small / UDP drops	Enlarge buffers before launching: `sudo sysctl -w net.core.rmem_max=16777216 net.core.rmem_default=8388608 net.core.netdev_max_backlog=50000`
`ros2 service call` no response	Incorrect service name (case mismatch)	Use `ros2 service list`; in v3.x the device model no longer appears in the topic/service path.

12. Technical Specifications

Category	Parameter	Specification
dToF Depth Module	Depth Resolution	240 × 180
	Range (Max)	25m @ 10% reflectance (<50Klux) / 50m @ 90% reflectance (<500lux)
	Range (Min)	0.15m
	Point Rate	Up to 430,000 pts/s
	Ranging Accuracy	±3 cm @ 1σ [1]
	FOV (H × V)	120° × 90°
	Angular Resolution	0.5° × 0.5°
	Laser Wavelength	940 nm
	Laser Safety Class	Class 1 eye-safe
	Frame Rate	Up to 10 FPS
RGB Camera Module	Resolution	1280 × 1088
	Frame Rate	30 FPS
	Exposure	Global Shutter
	FOV (H × V × D)	121° H × 106° V × 151° D
Overall	Built-in SLAM Localization Accuracy	±5 cm + 1% [2]
	Weight	274 g
	Dimensions (L×W×D)	Approx. 69 mm × 69 mm × 58 mm (body) / 66 mm (with connector)
	Power Consumption	Rated 11W, peak 24W
	Protection Rating	IP66
	Shock Resistance	50G × 5ms
	Operating Voltage	9–24 V
	Operating Temperature	-20°C ~ 45°C
	Storage Temperature	-20°C ~ 60°C
Data Output	Point Cloud	True-color SLAM / raw point cloud
	Photo	Raw RGB image
	IMU	High-frequency raw data
	Pose	High-frequency rotation and position data

[1] Indoor test environment, Std accuracy computed over 20 consecutive frames on an 80% Lambertian reflectance panel. ±3cm @ 1σ at 5m; 1σ accuracy < 1% at 5–30m.

[2] Laboratory conditions.

13. FAQ

❓ What operating systems does Odin1 Lite support?

Answer:

Ubuntu Version	Supported ROS Version
20.04	ROS1 Noetic
22.04	ROS2 Humble
24.04	ROS2 Jazzy

Supports x86 and ARM (Jetson) architectures.

❓ Device cannot be discovered or pinged after power-up?

Answer:

1. Confirm the device indicator light is steadily on (power is normal).
2. Confirm PC NIC IP and device are on the **same subnet**: - Device default IP: `192.168.1.251` - PC recommended setting: `192.168.1.x/24` (avoid conflict with device `.251`)
3. Use `ifconfig` to verify IP is actually active; if necessary, **disable and re-enable the NIC**.
4. Check whether the firewall blocks UDP broadcast (ports 60000–60010).
5. Device discovery, NIC selection, and UDP ports are handled by the SDK. Do not configure `device_ip` in launch arguments.

❓ How to assign an IP to the camera via DHCP?

Answer: Use the bundled `script/start_dhcp.sh` to start a udhcpd server:

1. Install udhcpd: `sudo apt install udhcpd`
2. Edit `script/my_dhcp.conf` to set the listening interface, IP pool, and gateway.
3. Start DHCP: `sudo ./script/start_dhcp.sh /path/to/my_dhcp.conf`
4. Check `/var/lib/misc/udhcpd.leases` to verify the camera obtained an IP.

❓ Multiple PCs on the same LAN, topics interfering with each other?

Answer:

- **ROS2**: Set a different `ROS_DOMAIN_ID` for each PC

`bash echo $ROS_DOMAIN_ID # Check current value (empty or 0 = shared) export ROS_DOMAIN_ID=1 # 1–99; 0 = shared`

- **ROS1**: Restrict master to localhost only, or use a different port

```bash # Recommended export ROS_LOCALHOST_ONLY=1

# Or: use a different port roscore -p 11312 export ROS_MASTER_URI=http://localhost:11312 ```

> The above commands apply only to the current terminal. Add them to `~/.bashrc` for permanent effect.

---

## ❓ Mapping / relocalization drifts or high packet loss?

**Answer:**

High-rate sensor data can overflow the kernel network receive buffers, causing UDP packet drops that affect SLAM and relocalization quality. Before running the driver, enlarge the buffers and set an isolated `ROS_DOMAIN_ID`:

```
# Enlarge kernel receive buffers
sudo sysctl -w net.core.rmem_max=16777216
sudo sysctl -w net.core.rmem_default=8388608
sudo sysctl -w net.core.netdev_max_backlog=50000

Isolate this PC from other ROS2 hosts on the same LAN
export ROS_DOMAIN_ID=1   # 1–99; use a different ID per PC
```

For permanent effect, add the `sysctl` settings to `/etc/sysctl.conf` (or a file under `/etc/sysctl.d/`) and add `export ROS_DOMAIN_ID=1` to `~/.bashrc`.

---

## ❓ `image_width` / `image_height` not taking effect under ROS1?

**Answer:** This was fixed in v0.9.0 and later. All v3.x drivers include the fix.

---

## ❓ Should I choose High Peak or Low Peak for Sensor Mode?

**Answer:**
```

Mode	Recommended Scenario	Description
0 / HDR	General	Auto-balances near and far
1 / High Peak (long frame)	**Outdoor**	Good strong-light rejection, slightly lower close-range accuracy
2 / Low Peak (short frame)	**Indoor**	Optimizes close-range and small-space noise, affected by strong light
3 / Adaptive	General	Adaptive switching

See 7. Quick Start §7.9 for switching instructions.

❓ Build fails with `ODIN SDK not found`?

Answer: Usually this happens when the `module/sdk_api` submodule is not entered on the first build. Try one of the following:

1. Use the `script/build_ros1.sh` / `script/build_ros2.sh` scripts included with the driver package (the scripts automatically build and install the SDK).
2. Manually initialize the SDK submodule: `git submodule update --init module/sdk_api`.
3. If the SDK is pre-installed at a custom path, set the environment variable: `export ODIN_SDK_PREFIX=/path/to/install`.

❓ Build freezes the system or exhausts RAM?

Answer: Heavy translation units such as `post_process_node` need 1.0–1.6 GB each. `build_ros1.sh` / `build_ros2.sh` already cap parallelism by available memory, but you can override:

```
ODIN_BUILD_JOBS=2 ./src/ros_driver/script/build_ros2.sh
# or
MAKEFLAGS="-j2 -l3" ./src/ros_driver/script/build_ros2.sh
```

On 8 GB or less, use `-j1` / `-j2`. You can also wrap the build in a cgroup:

```
systemd-run --user --scope -p MemoryHigh=5G -p CPUWeight=20 \
  ./src/ros_driver/script/build_ros2.sh
```

❓ RViz reports a TF transform failure and cannot display point clouds?

Answer: When source TF is too infrequent, a point-cloud timestamp can occur after the latest TF and tf2 cannot interpolate. Enable periodic TF republishing when launching the driver:

```
ros2 launch odin_ros_driver_rev1 odin1Lite.launch.py tf_publish_rate:=50
```

Use 50–200 Hz; `0` disables periodic republishing.

❓ Are high-rate topics dropped while recording ROS2 rosbag?

Answer: Record with the driver-provided QoS override file:

```
ros2 bag record -a \
  --qos-profile-overrides-path ~/catkin_ws/src/ros_driver/config/rosbag2_qos.yaml
```

❓ `ros2 topic hz` shows a low frame rate for `cloud_raw`. Is the driver publishing too slowly?

Answer: Not necessarily. Some users have reported that `ros2 topic hz` prints a noticeably low frequency for `cloud_raw`. This is usually caused by how `ros2 topic hz` measures the rate, especially after relocalization starts and the algorithm begins sending larger point-cloud data. `ros2 topic hz` counts messages on the Python subscriber side; for large `PointCloud2` messages it is affected by deserialization and transport overhead, so the displayed rate can be lower than the actual publish rate.

To verify the real frame rate, record a bag with the QoS override file and compute the rate from the message count:

```
ros2 bag record /manifold/device0/cloud/raw \
  --qos-profile-overrides-path ~/catkin_ws/src/odin_ros_driver/config/rosbag2_qos.yaml
```

After recording, use `ros2 bag info <bag>` to get `Message count` and `Duration`, then calculate `message_count / duration`.

> For real-time observation, launch the built-in `diag_subscriber_node` (C++ subscriber), which is more accurate than `ros2 topic hz` for large messages such as `PointCloud2`.

❓ Does `post_process_node` crash when it publishes the first image?

Answer: Multiple OpenCV installations can cause ABI or link-version mismatch with ROS `cv_bridge`. Rebuild `vision_opencv` / `cv_bridge` against the system OpenCV, then rebuild the driver.

❓ How to debug a crashing node with GDB?

Answer: Build in Debug mode first, then launch with GDB:

```
# Debug build
./src/ros_driver/script/build_ros2.sh --cmake-args -DCMAKE_BUILD_TYPE=Debug

# Interactive GDB
ros2 launch odin_ros_driver_rev1 odin1Lite.launch.py --launch-prefix "gdb -ex run --args"

# Non-interactive (good for SSH / embedded)
ros2 launch odin_ros_driver_rev1 odin1Lite.launch.py \
  --launch-prefix "gdb -batch -ex run -ex 'bt full' --args"

# Debug a single node only
ros2 launch odin_ros_driver_rev1 odin1Lite.launch.py \
  --launch-prefix "gdb -ex run --args" --launch-prefix-filter post_process_node
```

After the crash run `bt full` and `thread apply all bt` in GDB. If the stack shows `??`, rebuild with `-DCMAKE_BUILD_TYPE=Debug`.

❓ How can I confirm hardware JPEG decoding is active?

Answer: Startup logs only describe the compile-time decoding path. Confirm that hardware decode interrupts increase, `post_process_node` holds the hardware-decoder device node, and CPU use remains low.

❓ Crash on Jetson when hardware JPEG decode falls back to software?

Answer: On Jetson R36/JetPack 6, HW-to-SW JPEG fallback may crash due to a symbol clash between `libnvjpeg.so` and OpenCV's `libjpeg-turbo`. To use pure software decoding safely,

rebuild with:

```
# ROS2 example; for ROS1 add -DJPEG_PREFER_SOFTWARE_DECODER=ON to build_ros1.sh
colcon build --cmake-args -DJPEG_PREFER_SOFTWARE_DECODER=ON
```

The default `JPEG_PREFER_SOFTWARE_DECODER=OFF` uses hardware decoding; enable `ON` only when you need stable software decode or encounter the fallback crash.

❓ How to upgrade firmware remotely?

Answer: Visit `http://192.168.1.251:8080` → **Update** page, drag in the firmware package; the system will auto-install and reboot. See 8. Firmware Upgrade for details.

❓ How to export calibration files / relocalization maps / device logs?

Answer: All can be done via HTTP Dashboard:

- Calibration File Management: Upload / download `camera_calib_<SN>.yaml`
- Map File Download: Export relocalization map
- System Log Download: Package as `.tar.gz`
- Telemetry Recording: Export as CSV

---

14. PTP Time Synchronization

Odin1 Lite supports high-precision time synchronization based on **PTP (Precision Time Protocol, IEEE 1588)**, aligning the device system clock with the host (Host) clock to **within milliseconds**, so that hardware timestamps of LiDAR, camera and IMU data share the same time base as the host clock. This meets the strict time-consistency requirements of multi-sensor fusion and multi-robot coordination.

> By default, Odin1 Lite uses the system clock (SysTick) as the sensor data timestamp source. No configuration is required unless a PTP environment is deployed.

14.1 PTP Protocol Overview

14.1.1 Protocol Basics

PTP (IEEE 1588) is a high-precision clock synchronization protocol for local area networks. Compared with NTP (millisecond to ten-millisecond accuracy), PTP achieves **sub-microsecond accuracy** through **hardware timestamping**, and is widely used in industrial automation, telecom, automotive and robotics.

PTP uses a **Master-Slave architecture**:

- **Grandmaster (GM)**: the time source of the network, usually the host or a dedicated clock server;
- **Slave**: devices aligned to the master clock; Odin1 Lite acts as a slave in the synchronization network;
- **Boundary Clock (BC)** / **Transparent Clock (TC)**: intermediate network devices (such as PTP switches) that correct per-hop link delay and prevent accumulated errors over multiple forwarding hops.

14.1.2 Synchronization Mechanism (Delay Request/Response)

Four message types are exchanged between master and slave to measure clock offset and link delay:

```
Master Clock                                  Slave Clock
      |                                           |
      | ── Sync (hardware timestamped t1) ──────► | t2
      | ── Follow_Up (carries t1) ──────────────► |
      |                                           |
      | ◄────────── Delay_Req (timestamped t3) ── | t3
      | ◄──────── Delay_Resp (carries t4) ─────── | t4
      |                                           |
  offset = [(t2-t1) - (t4-t3)] / 2
  delay  = [(t2-t1) + (t4-t3)] / 2
```

1. The master periodically broadcasts `Sync` messages, which are **hardware timestamped** by the PHY/MAC when leaving the NIC (t1); the slave records its receive timestamp (t2);
2. The `Follow_Up` message conveys t1 to the slave;
3. The slave sends `Delay_Req` (timestamped t3) and the master replies `Delay_Resp` (carrying t4);
4. The slave computes the clock offset and path delay with the formulas above, and gradually **disciplines** its local clock.

The key to accuracy is that **t1–t4 are all produced by hardware**, eliminating the unpredictable software stack latency (jitter can reach tens of microseconds) — the core advantage of PTP over software-only schemes.

14.1.3 Transport: L2 (IEEE 802.3)

PTP messages can be carried over UDP/IPv4 (L3) or directly over Ethernet frames (L2, EtherType `0x88F7`). Odin1 Lite uses **L2 (IEEE 802.3)**:

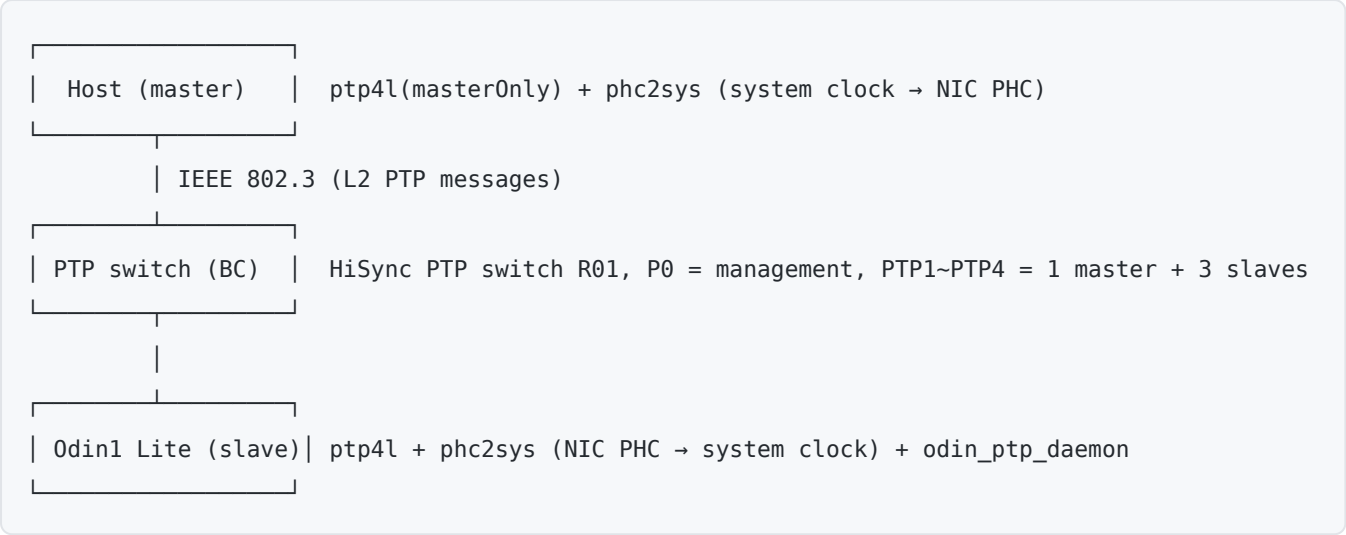
- L2 messages bypass the IP stack, giving lower latency and higher path determinism;
- All PTP ports in the network must use the same transport protocol (switch ports must be set to IEEE 802.3 in this solution).

14.1.4 Master Election (BMCA)

Clocks elect the master via the **Best Master Clock Algorithm (BMCA)** based on clockClass, clockAccuracy and Priority1/Priority2. In a typical Odin1 Lite deployment, the host is explicitly configured as `masterOnly`, and the switch acts as a boundary clock that synchronizes upstream to the host and provides time downstream to devices.

14.2 System Architecture

The Odin1 Lite PTP system consists of three layers:



Component	Role	Description
Host `ptp4l`	Master port	`-H` (hardware timestamp) `-2` (L2 transport) `--masterOnly=1`
Host `phc2sys`	Clock disciplining	Synchronizes the host system clock (CLOCK_REALTIME) to the NIC PHC
PTP switch	Boundary clock (BC)	Follows the host upstream, provides time to up to three devices downstream; SYNC LED visualizes sync state
Device `ptp4l` / `phc2sys`	Slave clock	Aligns the NIC PHC with the master and synchronizes PHC to the device system clock
Device `odin_ptp_daemon`	Resident daemon	Manages device-side PTP sessions together with the driver time-sync mode

14.3 Network and Switch Configuration

Odin1 Lite builds the synchronization network through a **HiSync PTP switch (model R01, Beijing Feiling)**:

Item	Description
Ports	`P0`: management PC (WebUI configuration, **not PTP-capable**); `PTP1 ~ PTP4`: 1 master + 3 slave PTP ports
WebUI	`http://192.168.1.82` (must be in the same subnet as Odin1 Lite / Host)
Login	user `admin` / password `admin`
LED	`SYNC`: slow green blink = PHC not synchronized; solid green = synced with upstream PHC

Configuration steps:

1. Connect the management PC to `P0`, set it to the `192.168.1.x` subnet, and open `http://192.168.1.82` in a browser;

2. Navigate to "Time Synchronization → PTP" in the left menu, then click "Port Configuration";
3. If the `Protocol` column of any PTP port is not **L2**, click "Edit", change Protocol to **IEEE 802.3** (keep other settings default) and save;
4. After **all** PTP ports are set to L2, click "Save & Apply" and wait for the switch to reboot.

> Both the master host and slave devices must connect to `PTP1 ~ PTP4`; do not use `P0`.

14.4 Host Configuration

Prerequisite: verify that the host NIC supports hardware timestamping (`ethtool -T <iface>` output contains `hardware-raw-clock` / `SOF_TIMESTAMPING_RAW_HARDWARE`), and identify the NIC connected to the switch (example below uses `enp1s0f0` — replace with your actual NIC).

```
# Start ptp4l: L2 transport + hardware timestamps + master-only
sudo ptp4l -i enp1s0f0 -m -H -2 --masterOnly=1

# Start phc2sys: synchronize system clock to NIC PHC (in another terminal)
sudo phc2sys -s CLOCK_REALTIME -c enp1s0f0 -m -O 0 -S 0.001
```

After the host is started, the switch should synchronize to the host regardless of whether the device is connected — verify via the switch `SYNC` LED (solid green) or the WebUI "Sync Status" page (sync status, PTP clock time, offset, grandmaster info, etc.).

14.5 Hardware Support Check and Software-Only Synchronization

14.5.1 Checking Whether the Host NIC Supports PTP Hardware Timestamping

PTP hardware timestamping relies on the NIC's PHC (PTP Hardware Clock). Run the following checks on the host:

① Check NIC timestamping capability

```
ethtool --json -T enp1s0f0     # or legacy: ethtool -T enp1s0f0
```

Output Key	Meaning	Conclusion
`hardware-raw-clock: true` / `SOF_TIMESTAMPING_RAW_HARDWARE` with `ptp-hardware-clock` pointing to `/dev/ptpN`	NIC has a dedicated PHC	✅ Hardware sync supported, deploy as in 14.4
Only `software-raw-clock: true` / `SOF_TIMESTAMPING_SOFTWARE`	No PHC	❌ Not supported, see 14.5.2 software-only sync

NICs commonly supporting hardware timestamping: Intel i210/i211/i225/i226, 82574, 82599/X520, X550 series. Most consumer onboard NICs (e.g. Realtek RTL8111/8125, some Intel 1225-V variants) **do not** support it.

② Check the PHC device node

```
ls /dev/ptp*
# NICs with hardware timestamping expose /dev/ptp0 etc.;
# match via the ptp-hardware-clock field in ethtool --json -T output
```

③ Verify the PHC is readable

```
sudo phc_ctl /dev/ptp0 get
```

If the current PHC time is returned, the hardware clock works.

> The device side (Odin1 Lite) already has a built-in PHC; the checks above target the Host only. If the host lacks hardware timestamping, the whole chain degrades to software-sync accuracy.

14.5.2 Software Timestamping (No PHC on Host)

If the host NIC lacks hardware timestamping, PTP can still run in linuxptp's **software timestamping** mode: the timestamping point falls back from PHY/MAC to the kernel network stack. User-space latency is eliminated, but kernel transmit/receive path jitter remains — typical accuracy degrades from sub-microsecond to **about 10 µs ~ 1 ms**.

Only one change is needed on the host (compared with 14.4): `-H` (hardware timestamps) becomes `-S` (software timestamps):

```
# Host: software-timestamped master (-S = software timestamping)
sudo ptp4l -i enp1s0f0 -m -S -2 --masterOnly=1

# phc2sys no longer applies (no PHC on host):
# in -S mode ptp4l timestamps come directly from CLOCK_REALTIME,
# so the host system clock itself is the published time base

# Device side: keep hardware-timestamped slave mode unchanged (odin_ptp_daemon / ptp4l -H)
```

Note: the device-side PHC is always available, so keep hardware timestamping on the device; the overall chain accuracy is limited by host software timestamps. If higher accuracy is required without a PHC-capable NIC, consider installing one (e.g. Intel i226 — inexpensive and driver-free on mainstream distros).

Accuracy and selection guide

Option	Host requirement	Typical accuracy	Use case
Hardware PTP (recommended)	NIC with PHC	Sub-microsecond	Tight multi-sensor fusion, motion reconstruction
Software PTP (-S)	Any NIC	10 µs ~ 1 ms	Coarse time alignment, log correlation

14.5.3 Alternative PTP Implementations for Software Sync

Software sync has no NIC hardware requirement, and the implementation is not limited to linuxptp. Besides the default **linuxptp (ptp4l)**, the mainstream alternatives are:

Tool	Protocol/Profile	Platform	Notes	Suitability
linuxptp (ptp4l + phc2sys)	IEEE 1588-2019 full profiles (incl. L2/UDP, 802.1AS)	Linux	Official kernel companion, most complete, supports both hardware and software timestamping	✅ Recommended, default in this document
ptpd2 (PTPd)	IEEE 1588-2008	Linux/BSD	Legacy implementation, simple config, declining community activity	⚠️ Usable, no new features, not recommended for new projects
OpenAvnu gPTP	IEEE 802.1AS (gPTP)	Linux/Windows	Intel-led, targeting AVB/TSN audio-video scenarios, available on Windows	⚠️ Only for gPTP/TSN scenarios

Notes:

1. **802.1AS (gPTP)**: for automotive/AVB/TSN networks using the gPTP profile, confirm the PTP switch also supports 802.1AS and that the device-side daemon is compatible — contact Manifold technical support first;
2. **ptpd2 and ptp4l must not run simultaneously** on the same host (port conflict);
3. Software-sync accuracy is limited by the **timestamping point** (kernel network stack), not the tool; all tools show essentially the same accuracy in software-timestamp mode.

14.6 Device Configuration

Device-side PTP sessions are automatically managed by the firmware-resident `odin_ptp_daemon` (together with `ptp4l` / `phc2sys`). **No manual login to the device is required.**

All user operations are done from the host via ROS services:

```
# Query current sync mode (0=SysTick system clock, 1=PTP)
ros2 service call /manifold/device0/get_sync_mode \
  odin_ros_driver_rev1/srv/GetSyncMode "{}"

# Switch to PTP hardware timestamp mode (takes effect after device reboot)
ros2 service call /manifold/device0/set_sync_mode \
  odin_ros_driver_rev1/srv/SetSyncMode "{sync_mode: 1}"
```

Once PTP is active, query the sync quality in real time:

```
ros2 service call /manifold/device0/get_ptp_sync_status \
  odin_ros_driver_rev1/srv/GetPtpSyncStatus "{}"
```

The returned snapshot includes sync status, offset from master, mean path delay and frequency adjustment — useful for quantitatively evaluating sync accuracy.

> If `get_ptp_sync_status` keeps reporting unsynchronized, check in order: switch port protocol is L2 (14.3) → host master clock is running (14.4) → device sync mode is PTP (14.6); if still unresolved, contact Manifold technical support.

14.7 Sync Verification

1. **Switch sync status**: `SYNC` LED solid green; WebUI "Sync Status" shows synchronized with offset within milliseconds;
2. **PTP status query**: confirm device-side sync status and offset via `get_ptp_sync_status` (see 14.6);

3. **Data timestamp check**: after launching the driver on the host, verify sensor topic timestamps use the synchronized clock:

```
ros2 launch odin_ros_driver_rev1 odin1Lite.launch.py
ros2 topic echo /manifold/device0/imu --field header.stamp
```

14.8 Notes

1. All PTP ports in the network must uniformly use **L2 (IEEE 802.3)**; mixing UDPv4 prevents synchronization. Software-sync mode also uses L2, so the switch port requirement is unchanged;
2. `set_sync_mode` requires a **device reboot** to take effect;
3. The device-side `odin_ptp_daemon` is mutually exclusive with the system default `ptp4l` / `phc2sys`; running both causes sync anomalies;
4. The host NIC should support hardware timestamping; otherwise fall back to software timestamps (`ptp4l -S`) with significantly reduced accuracy (see 14.5);
5. The switch, host and devices must be in the same subnet: host and devices connect to switch `PTP1 ~ PTP4`, management PC to `P0`;
6. Monitor the offset via `get_ptp_sync_status` periodically; if it keeps growing, check the link, switch port configuration and host master clock first.

← Previous: FAQ | Back to Home →