

Nodar SDK 2.12.2

None

Table of contents

1. SDK	3
1.1 Supported Systems	3
1.2 Installation	4
1.3 Multiple Versions	5
1.4 Licenses and Activation Keys	5
1.5 Getting Started	5
2. ROS2	9
2.1 Hammerhead ROS2	9
2.2 Python Examples	15
2.3 C++ Examples	24
3. ZMQ	33
3.1 Hammerhead ZMQ	33
3.2 Python Examples	40
3.3 C++ Examples	60
4. Frequently Asked Questions	87

1. SDK

Download Now

If you are looking for a full hardware solution, take a look at our preconfigured Hardware Development Kits (HDKs), which include access to this SDK as well as all the hardware necessary to run Hammerhead out of the box.

If you want to use your own compute device and/or cameras, then you'll need an SDK license. This gives you access to our Ubuntu `.deb` packages containing our software binaries:

- `hammerhead`
- `nodar_viewer`

You'll also get access to our extensive library of C++ and Python examples that you can use as a starting point for your own integration:



ZMQ Code

ZMQ Docs



ROS2 Code

ROS2 Docs

Ready to go?

- buy.nodarsensor.net for Free Trials and Purchases
- sales@nodarsensor.com for Questions

Download Offline PDF

1.1 Supported Systems

Our binaries rely on Cuda and currently target Ubuntu 20.04 and Ubuntu 22.04 for ARM and AMD64 (Intel and AMD Cpus). Here is a list of configurations we currently provide `.deb` packages for:

- Cuda 11.4 / Ubuntu 20.04 / Amd 64
- Cuda 11.4 / Ubuntu 20.04 / Arm 64
- Cuda 12.0 / Ubuntu 20.04 / Amd 64
- Cuda 12.1 / Ubuntu 22.04 / Amd 64
- Cuda 12.2 / Ubuntu 22.04 / Amd 64
- Cuda 12.2 / Ubuntu 22.04 / Arm 64
- Cuda 12.3 / Ubuntu 22.04 / Amd 64
- Cuda 12.6 / Ubuntu 22.04 / Amd 64
- Cuda 12.6 / Ubuntu 22.04 / Arm 64

If there is something that you would like us to support, please let us know at sales@nodarsensor.com

1.2 Installation

After purchasing a free trial or an extended license through buy.nodarsensor.net, you will get an email with:

- Activation key (like `ABCDE-ABCDE-ABCDE-ABCDE`)
- Activation link (like `https://buy.nodarsensor.net/activate.html?token=random_token`)

After clicking the activation link, you will get another email with a custom download link like:

- `https://downloads.nodarsensor.net/abcdefgh-1234-abcd-1234-abcdefghijkl`
(This link is a dummy link that **will not work**)

This link is your custom portal for downloading `.deb` packages.

The portion of the link after `downloads.nodarsensor.net` is your customer UUID.

In this example, `abcdefgh-1234-abcd-1234-abcdefghijkl`

With this download URL in hand, you can either download the `.deb` packages directly, or let our `nodar-quickstart.sh` script do the hard part, and find the best debs for your system. Just enter your customer UUID and whichever options you want:

Download nodar-quickstart.sh

Note: After downloading, you should make `nodar-quickstart.sh` executable:

```
chmod +x nodar-quickstart.sh
```

```
./nodar-quickstart abcdefgh-1234-abcd-1234-abcdefghijkl
./nodar-quickstart abcdefgh-1234-abcd-1234-abcdefghijkl --download
./nodar-quickstart abcdefgh-1234-abcd-1234-abcdefghijkl --install
```

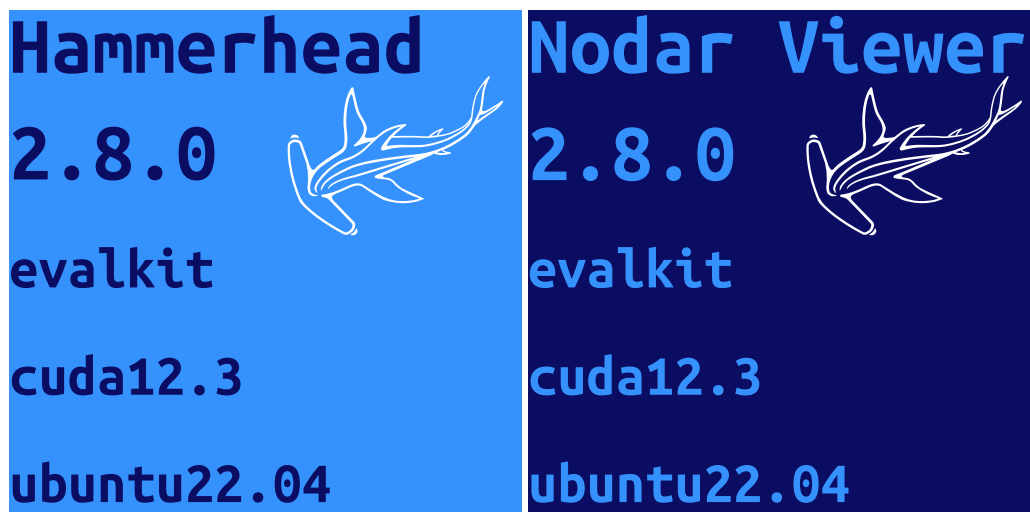
By default, our downloads will save to `${HOME}/.config/nodar/`. For example, depending on your system, the `--download` or `--install` option in `nodar-quickstart.sh` might download the following files:

```
~$ ls ~/.config/nodar/downloads/debs/
hammerhead-2.12.1-evalkit-gd-cuda12.3-ubuntu22.04-amd64.deb
nodar_viewer-2.12.1-evalkit-gd-cuda12.3-ubuntu22.04-amd64.deb
```

The `--install` option will install these packages automatically. If you want to do that manually instead, then you can

```
cd ~/.config/nodar/downloads/debs/
sudo apt install \
  ./hammerhead-2.12.1-evalkit-gd-cuda12.3-ubuntu22.04-amd64.deb \
  ./nodar_viewer-2.12.1-evalkit-gd-cuda12.3-ubuntu22.04-amd64.deb
```

After installation, `hammerhead` and `nodar_viewer` will be available on your system. We also install unique desktop icons on Ubuntu to make it easier to launch:



1.3 Multiple Versions

To make upgrading and downgrading as easy as possible, we support parallel installations. For example, the following is valid:

```
sudo apt install ./hammerhead-2.8.0-evalkit-cuda12.3-ubuntu22.04-amd64.deb
sudo apt install ./hammerhead-2.9.0-evalkit-cuda12.3-ubuntu22.04-amd64.deb
```

When you install one or more versions, they are automatically added to your `update-alternatives`. This means that you can change which version `hammerhead` and `nodar_viewer` point to by configuring the `update-alternative`:

```
~$ sudo update-alternatives --config hammerhead
There are 2 choices for the alternative hammerhead (providing /usr/bin/hammerhead).

  Selection    Path
  -----
0            /usr/lib/nodar/2.8.0-evalkit-cuda12.3-ubuntu22.04-amd64/hammerhead/nodar/hammerhead
* 1            /usr/lib/nodar/2.9.0-evalkit-cuda12.3-ubuntu22.04-amd64/hammerhead/nodar/hammerhead
2            /usr/lib/nodar/2.8.0-evalkit-cuda12.3-ubuntu22.04-amd64/hammerhead/nodar/hammerhead

Priority      Status
-----
22812302     auto mode
22912302     manual mode
22812302     manual mode

Press <enter> to keep the current choice[*], or type selection number:

matt@desk:~/Desktop/hammerhead$ sudo update-alternatives --config nodar_viewer
There are 2 choices for the alternative nodar_viewer (providing /usr/bin/nodar_viewer).

  Selection    Path
  -----
0            /usr/lib/nodar/2.8.0-evalkit-cuda12.3-ubuntu22.04-amd64/nodar_viewer/nodar/nodar_viewer
* 1            /usr/lib/nodar/2.9.0-evalkit-cuda12.3-ubuntu22.04-amd64/nodar_viewer/nodar/nodar_viewer
2            /usr/lib/nodar/2.8.0-evalkit-cuda12.3-ubuntu22.04-amd64/nodar_viewer/nodar/nodar_viewer

Priority      Status
-----
22812302     auto mode
22912302     manual mode
22812302     manual mode

Press <enter> to keep the current choice[*], or type selection number:
```

If you are unsure of which version you have installed, you can always pass the `--version` flag to `hammerhead` and `nodar_viewer`:

```
matt@desk:~$ hammerhead --version
Hammerhead 2.12.1 Evalkit_gd
Commit d1ddd8c2-R
Compiled on:
  2025-12-04 at 17:29:35 UTC
  Ubuntu 22.04.3 LTS
  Cuda 12.3.103 for compute architecture(s): 86-real, 87-real, 89-real

matt@desk:~$ nodar_viewer --version
Hammerhead 2.12.1 Evalkit_gd
Commit d1ddd8c2-R
Compiled on:
  2025-12-04 at 17:29:35 UTC
  Ubuntu 22.04.3 LTS
  Cuda 12.3.103 for compute architecture(s): 86-real, 87-real, 89-real
```

1.4 Licenses and Activation Keys

Licenses are **node-locked**, meaning they are tied to a machine's hardware and **cannot be transferred** to another machine.

When you start a trial or purchase a license, you receive one activation key that is valid for a specific number of machines. For example, a 3-machine license includes a single activation key that can be used simultaneously on all three machines.

Since the licenses are bound to hardware, not the operating system, you can use the same license in different Docker containers on the same machine. These only count as one machine against your quota unless the Docker image is intentionally hiding hardware details.

Still in doubt?

Get a Free Trial

1.5 Getting Started

1.5.1 Quick Start

The quickest way to get up and running is with [nodar-quickstart.sh](#).

Download nodar-quickstart.sh

***Note:** After downloading, you should make `nodar-quickstart.sh` executable:

```
chmod +x nodar-quickstart.sh
```

```
./nodar-quickstart.sh <uuid or download link> --run
```

This will

- Install the `.deb` packages
- Install a very minimal dataset to `${HOME}/.config/nodar/downloads/datasets`
- Launch `nodar_viewer`
- Launch `hammerhead` to run on the dataset

If you want to manually get the dataset, you can download it here:

`nodar_data_day_highway_01_minimal.zip`

After that, all you need to do is to click `Connect` in `nodar_viewer` :

On your first run, you will be asked to enter your activation key that you received by email (something like `ABCDE-ABCDE-ABCDE-ABCDE`).

- `-c` : sets the path for the configuration files
- `-s` : sets the path for the images used for playback
- `-l` : tells Hammerhead to loop over the playback data

You can visualize the output by opening `nodar_viewer` in another terminal and clicking `Connect`:

1.5.2 Playback from Other Sources

`nodar-quickstart.sh` plays back data from file. Specifically, it invokes `hammerhead` with a command like:

```
hammerhead -c "config/" -s "topbot/"
```

which sets the path to the configuration files and to a folder containing topbot images.

To playback directly from cameras, you will omit the `-s` option and instead modify the `master_config.ini` file. At the moment we support 2 options:

1. Direct readout of Lucid cameras
2. Reading through a network interface

Lucid Cameras

If you have Lucid cameras that you want to use, then you should modify the `master_config.ini` with your camera IDs:

```
camera_source = lucid

# [string] Camera serial numbers specifying left and right camera
# Camera serial numbers can be found on the label attached to camera body
camera_left_id = 231001324
camera_right_id = 231001327
```

and call `hammerhead` **without** the `-s` option:

```
hammerhead -c "config/"
```


Topbot Publisher

If you want to use other types of cameras, then you will be in charge of synchronizing them. You should then modify the `master_config.ini` with the IP address and port of your topbot publisher, and specify the dimensions of the images you will be providing:

```
# [string] Specify camera source
# This can be either 'lucid' or 'zmq://<ipv4>:<port>'
# If empty we assume that you are using 'lucid'
# An example of a valid zmq camera source is 'zmq://127.0.0.1:5000'
camera_source = zmq://127.0.0.1:5000

# [string] Specify width and height for zmq camera source
# This is only used if camera_source is set to zmq://<ipv4>:<port>
# This is not used if camera_source is set to lucid
zmq_camera_source_width = 2880
zmq_camera_source_height = 1860
```

As with the Lucid cameras, you would then call `hammerhead` without the `-s` option:

```
hammerhead -c "config/"
```

These 2 examples describe how to use the Topbot Publisher in much more detail:

C++ ZMQ Topbot Publisher

Python ZMQ Topbot Publisher

1.5.3 Configuration Files

Both `hammerhead` and `nodar_viewer` can be used on the command line. Almost everything you need to run those programs is packaged in the `.deb` files, with 3 notable exceptions:

- `extrinsics.ini`
- `intrinsics.ini`
- `master_config.ini`

These files will be unique to your camera system, so you must create them yourself. As the names imply, they need to contain the extrinsics, intrinsics, and high-level configuration settings for how you want to run Hammerhead. You can find extensive documentation for these files here:

<https://nodarsensor.notion.site/documentation>

There are 2 primary ways to pass these files to Hammerhead:

1. Pass the directory with these configuration files as a command-line option:

```
hammerhead -c config/
```

where the `config/` looks like:

```
config/
├── extrinsics.ini
├── intrinsics.ini
└── master_config.ini
```

1. Save the config files in your home directory, specifically, `${HOME}/.config/nodar/config/`:

```
${HOME}/.config/nodar/config/
├── extrinsics.ini
├── intrinsics.ini
└── master_config.ini
```

and run `hammerhead` **without** the `-c` option.

Typically, you will use Option 1. if you are experimenting with different configurations, and Option 2. once you have finalized your configuration.

`nodar_viewer` doesn't use these config files. You can (and should) run it with no options.

1.5.4 Images

There are 3 primary ways to pass images to Hammerhead:

1. Use the `-s` option to point to a directory containing sequentially-numbered topbot images.

`topbot` == image where a synchronized left and right image are stacked vertically (left on top, right on bottom)

1. Use Lucid cameras and provide their IDs in the `master_config.ini` (`camera_left_id` and `camera_right_id`).
2. Publish topbot images using one of our communication interfaces (we recommend ZMQ). See our [C++](#) and [Python](#) examples for more details.

2. ROS2

2.1 Hammerhead ROS2

A comprehensive ROS2 client library for interfacing with the Hammerhead stereo vision system



2.1.1 Table of Contents


- [Overview](#)
- [Quick Start](#)
- [Message Types & Topics](#)
- [Project Structure](#)
- [Examples & Tutorials](#)
- [3D Coordinate System & Point Cloud Conversion](#)
- [API Reference](#)
- [Best Practices & Tips](#)

2.1.2 Overview

The Hammerhead system is a high-performance stereo vision processing unit that publishes various types of data over ROS2. This library provides easy-to-use APIs for receiving and processing:

- **Stereo Images** - Raw and rectified left/right camera feeds
- **Depth Data** - Disparity maps and color-blended depth images
- **Point Clouds** - 3D point cloud data with or without RGB information
- **Obstacle Detection** - Real-time obstacle data with bounding boxes
- **Camera Control** - Parameter adjustment and recording control

Key Features

Feature	Description
 Python Ready	Complete Python packages with examples and utilities
 High Performance C++	Optimized C++ implementation for real-time applications
 ROS2 Native	Full ROS2 integration with standard message types

2.1.3 Quick Start

Repository Setup

```
# Get the Hammerhead ROS2 repository
git clone git@github.com:nodarhub/hammerhead_ros2.git

# If you received the HDK with version X.X.X, you can check out the corresponding tag (skip this step if you want the latest version):
git checkout X.X.X
```



```
# Make sure that the submodules are up to date
git submodule update --init --recursive
```

ROS2 Installation & Usage

```
# Build the workspace
cd hammerhead_ros2
colcon build

# Source the workspace
source install/setup.bash

# View Live left raw image from Hammerhead
ros2 run image_viewer image_viewer /nodar/left/image_raw

# Generate point cloud data and save to rosbag
ros2 run generate_rosbag2 generate_rosbag2

# Record obstacle detection data
ros2 run obstacle_data_recorder obstacle_data_recorder
```

Build Scripts

The convenience scripts `compile.sh` and `clean.sh` build and clean all the examples while making sure that all the build artifacts always remain in the same place.

```
# Build all examples
./compile.sh

# Clean build artifacts
./clean.sh
```

2.1.4 Message Types & Topics

Hammerhead publishes data using standard ROS2 message types over predefined topics:

Image Topics

Topic	Description	Message Type
<code>/nodar/left/image_raw</code>	Raw left camera feed	<code>sensor_msgs/Image</code>
<code>/nodar/right/image_raw</code>	Raw right camera feed	<code>sensor_msgs/Image</code>
<code>/nodar/left/image_rect</code>	Rectified left image	<code>sensor_msgs/Image</code>
<code>/nodar/right/image_rect</code>	Rectified right image	<code>sensor_msgs/Image</code>
<code>/nodar/disparity</code>	Disparity map (Q12.4 format)	<code>sensor_msgs/Image</code>
<code>/nodar/color_blended_depth/image_raw</code>	Color-coded depth visualization	<code>sensor_msgs/Image</code>

3D Data Topics

Topic	Description	Message Type
<code>/nodar/point_cloud</code>	3D point cloud data	<code>sensor_msgs/PointCloud2</code>
<code>/nodar/obstacle</code>	Obstacle detection data	<code>hammerhead_msgs/ObstacleData</code>

Control Topics

Topic	Description	Message Type
<code>/nodar/camera_param</code>	Camera parameter control	<code>hammerhead_msgs/CameraParam</code>
<code>/nodar/recording</code>	Recording on/off control	<code>std_msgs/Bool</code>

2.1.5 Project Structure

The `hammerhead_msgs` folder contains custom message definitions for Hammerhead-specific data types like obstacle detection and camera parameters.

The `examples` folder contains comprehensive examples that demonstrate how to interact with Hammerhead using ROS2. We envision that you will use these examples as a jumping-off point for your application.

We suggest that you start by examining the code and README's in the individual example directories for more details about what each example does.

2.1.6 Examples & Tutorials

Python Examples

Python examples provide easy-to-use scripts for common Hammerhead integration tasks.

VISUALIZATION EXAMPLES

- **Image Viewer** - Real-time OpenCV viewer for stereo images, disparity maps, and depth data

DATA GENERATION EXAMPLES

- **Generate ROS Bag** - Generate point cloud data and save to ROS2 bag files
- **Point Cloud Generator** - Generate 3D point clouds from stereo data
- **Obstacle Data Recorder** - Record obstacle detection data

CONTROL EXAMPLES

- **Camera Parameter Control** - Real-time camera parameter adjustment

C++ Examples

High-performance C++ implementations for real-time applications and system integration.

VISUALIZATION EXAMPLES

- **Image Viewer** - Real-time OpenCV viewer for stereo images, disparity maps, and depth data

DATA GENERATION EXAMPLES

- **Generate ROS Bag** - Generate point cloud data and save to ROS2 bag files
- **Point Cloud Generator** - Generate 3D point clouds from stereo data
- **Obstacle Data Recorder** - Record obstacle detection data

CONTROL EXAMPLES

- **Camera Parameter Control** - Real-time camera parameter adjustment

Common Integration Workflows

IMAGE PROCESSING PIPELINE

1. Start with **Image Viewer** to verify camera feeds
2. Use **Generate ROS Bag** to capture datasets
3. Process images with custom algorithms

3D RECONSTRUCTION WORKFLOW

1. Subscribe to point cloud topics to get 3D data
2. Use **Point Cloud Generator** to create custom point clouds
3. Process with 3D algorithms

4. Integrate with navigation or mapping frameworks

OBSTACLE DETECTION INTEGRATION

1. Use **Obstacle Data Recorder** to understand data format
2. Implement real-time processing of obstacle messages
3. Integrate with path planning or control systems
4. Add custom filtering or tracking algorithms

2.1.7 3D Coordinate System & Point Cloud Conversion

Hammerhead follows standard stereo reconstruction principles for converting disparity to 3D point clouds:

Disparity Scaling

The disparity is in Q12.4 format. We scale the disparity by `1 / 16.0` to get the disparity in `float32` format:

```
disparity_scaled = disparity.astype(np.float32) / 16.0
```

3D Reprojection

The scaled disparity map is reprojected into 3D space using OpenCV's `cv2.reprojectImageTo3D()` and a 4×4 reprojection matrix `Q`:

```
# Important: Negate the last row for correct coordinate frame
Q_corrected = Q.copy()
Q_corrected[3, :] = -Q_corrected[3, :]

# Reproject to 3D
points_3d = cv2.reprojectImageTo3D(disparity_scaled, Q_corrected)
```

A negative translation vector ($T_x < 0$) is used when creating the `Q` matrix to conform to the definition in OpenCV. This ensures that the point cloud is generated in a consistent right-handed coordinate frame. As a result, the entire last row of `Q` must be negated before passing to the `cv2.reprojectImageTo3D()` call.

This conversion scheme has been used in the following examples:

- **Generate ROS Bag** - C++ point cloud generation
- **Point Cloud Generator** - C++ real-time point cloud processing
- **Generate ROS Bag** - Python point cloud generation
- **Point Cloud Generator** - Python real-time point cloud processing

2.1.8 API Reference

Message Types

All Hammerhead ROS2 messages use standard ROS2 message types where possible, with custom messages defined in `hammerhead_msgs`.

STANDARD IMAGE MESSAGES

Used for all image data including raw stereo images, rectified images, and disparity maps.

```
#include <sensor_msgs/msg/image.hpp>
#include <rclcpp/rclcpp.hpp>

class ImageSubscriber : public rclcpp::Node
{
public:
    ImageSubscriber() : Node("image_subscriber")
    {
        subscription_ = this->create_subscription<sensor_msgs::msg::Image>(
            "/nodar/left/image_raw", 10,
            std::bind(&ImageSubscriber::image_callback, this, std::placeholders::_1));
    }
}
```



```
private:
    void image_callback(const sensor_msgs::msg::Image::SharedPtr msg)
    {
        // Process image data
        RCLCPP_INFO(this->get_logger(), "Received image: %dx%d", msg->width, msg->height);
    }

    rclcpp::Subscription<sensor_msgs::msg::Image>::SharedPtr subscription_;
};
```

OBSTACLEDATA

Contains real-time obstacle detection information with bounding boxes and velocity vectors.

```
#include <hammerhead_msgs/msg/obstacle_data.hpp>
#include <rclcpp/rclcpp.hpp>

class ObstacleSubscriber : public rclcpp::Node
{
public:
    ObstacleSubscriber() : Node("obstacle_subscriber")
    {
        subscription_ = this->create_subscription<hammerhead_msgs::msg::ObstacleData>(
            "/nodar/obstacle", 10,
            std::bind(&ObstacleSubscriber::obstacle_callback, this, std::placeholders::_1));
    }

private:
    void obstacle_callback(const hammerhead_msgs::msg::ObstacleData::SharedPtr msg)
    {
        // Process obstacle data
        for (const auto& obstacle : msg->obstacles) {
            RCLCPP_INFO(this->get_logger(), "Obstacle detected with %zu points",
                obstacle.bounding_box.points.size());
        }
    }

    rclcpp::Subscription<hammerhead_msgs::msg::ObstacleData>::SharedPtr subscription_;
};
```

2.1.9 Best Practices & Tips

Performance

- Use C++ for real-time applications
- Consider message buffering for high-frequency data
- Monitor system resources with large point clouds
- Use appropriate QoS settings for your application

Reliability

- Always validate message types and versions
- Implement proper error handling
- Use ROS2 lifecycle nodes for complex applications
- Add logging for debugging

ROS2 Integration

- Follow ROS2 naming conventions
- Use appropriate QoS policies
- Integrate with standard ROS2 tools (rviz2, rqt)
- Consider using composition for performance

Debugging

- Start with simple subscribers before custom code
- Use ROS2 command-line tools for inspection

- Check topic types and message frequencies
- Use ROS2 debugging tools and visualization

2.2 Python Examples

2.2.1 Generate ROS2 Bag

Convert data recorded by Hammerhead into ROS2 bag format for analysis and replay.

Build

```
cd hammerhead_ros2
colcon build --packages-up-to generate_rosbag2_py
```

Available Tools

xyz - POINT CLOUD ONLY

Generates a bag containing only `sensor_msgs/PointCloud2` messages with points containing `x,y,z` and `r,g,b` attributes.

everything - COMPLETE DATASET

Generates a bag containing the aforementioned point clouds as well as:

- Raw left and right camera images
- Rectified left camera image

Prerequisites

You need data recorded by Hammerhead in the following format:

```
20230208-133746/
disparity/
  000000000.tiff
  000000001.tiff
  ...
details/
  000000000.csv
  000000001.csv
  ...
```

Usage

```
# Source the workspace
source install/setup.bash

# Generate point cloud only bag
ros2 run generate_rosbag2_py xyz <recorded_data_directory>

# Generate complete dataset bag
ros2 run generate_rosbag2_py everything <recorded_data_directory>
```

EXAMPLES

```
# Generate complete bag from recorded data
ros2 run generate_rosbag2_py everything 20230208-133746

# Save bag to custom location
ros2 run generate_rosbag2_py everything 20230208-133746 ~/Downloads/my_bag
```

Output

The tool generates a ROS2 bag in the specified directory:

```
20230208-133746/
bag/
  bag_0.db3
  metadata.yaml
disparity/
```



```
000000000.tiff
...
details/
000000000.csv
...
```

Generated Topics

POINT CLOUD TOPICS

- `/nodar/point_cloud` - Generated 3D point cloud data

IMAGE TOPICS (WITH `everything` MODE)

- `/nodar/left/image_raw` - Raw left camera images
- `/nodar/right/image_raw` - Raw right camera images
- `/nodar/left/image_rect` - Rectified left camera images

Features

- Converts disparity data to 3D point clouds using standard stereo reconstruction
- Preserves timestamps from the original recording
- Generates standard ROS2 message types
- Configurable output location

Troubleshooting

- **Memory issues:** Use `xyz` mode for large datasets to reduce memory usage
- **Invalid path:** Check that the recorded data directory exists and contains the expected structure
- **Build errors:** Ensure all dependencies are installed and workspace is sourced

2.2.2 Image Viewer

Real-time OpenCV viewer for stereo images, disparity maps, and depth data published by Hammerhead.

Build

```
cd hammerhead_ros2
colcon build --packages-up-to image_viewer_py
```

Usage

```
# Source the workspace
source install/setup.bash

ros2 run image_viewer_py image_viewer_py <image_topic>
```

PARAMETERS

- **image_topic** : ROS2 topic name that provides `sensor_msgs::msg::Image` messages

EXAMPLES

```
# Source the workspace
source install/setup.bash

# View raw left camera
ros2 run image_viewer_py image_viewer_py /nodar/left/image_raw

# View disparity map
ros2 run image_viewer_py image_viewer_py /nodar/disparity
```

Available Image Topics

Topic	Description
<code>/nodar/left/image_raw</code>	Raw left camera
<code>/nodar/right/image_raw</code>	Raw right camera
<code>/nodar/left/image_rect</code>	Rectified left image
<code>/nodar/right/image_rect</code>	Rectified right image
<code>/nodar/disparity</code>	Disparity map
<code>/nodar/color_blended_depth/image_raw</code>	Color-coded depth visualization

Features

- Support for all image topic types
- Real-time display with OpenCV

Integration with ROS2 Tools

There is nothing special about the image topics published by Hammerhead. You can also view them with tools like `rviz2` and `rqt`.

Troubleshooting

- **No display appears**: Check that Hammerhead is running and publishing image topics
- **Topic not found**: Verify the topic name using `ros2 topic list`
- **Build errors**: Ensure all dependencies are installed and workspace is sourced

Press **Ctrl+C** to exit the viewer.

2.2.3 Obstacle Data Recorder

A recorder for obstacle data published by Hammerhead.

Overview

This example demonstrates how to subscribe to `hammerhead_msgs/ObstacleData` messages from Hammerhead and save the detection data in structured text files for offline analysis and processing. The obstacle data is represented in the XZ plane (bird's eye view), where each obstacle is defined by:

- **Bounding box:** Collection of 3D points defining the obstacle perimeter
- **Velocity vector:** Movement direction and speed (currently in development)

Build

```
cd hammerhead_ros2
colcon build --packages-up-to obstacle_data_recorder_py
```

Usage

```
# Source the workspace
source install/setup.bash

# Run the obstacle data recorder
ros2 run obstacle_data_recorder_py obstacle_data_recorder_py
```

Features

- **Real-time Recording:** Subscribes to live obstacle detection data
- **Batch Processing:** Handles multiple obstacles per detection frame
- **Timestamped Data:** Preserves timing information for analysis

Topic Interface

SUBSCRIBED TOPICS

- `/nodar/obstacle_data` - Obstacle detection messages from Hammerhead

Output Format

The recorder creates an `obstacle_data` folder containing timestamped text files:

```
obstacle_data/
  obstacle_YYYYMMDD_HHMMSS_001.txt
  obstacle_YYYYMMDD_HHMMSS_002.txt
  ...
```

Each file contains:

- **Header:** Parameter descriptions and data format
- **Data:** Obstacle bounding box points and velocity vectors

DATA STRUCTURE

Each obstacle includes:

- **Bounding box points:** 3D coordinates (X, Z plane)
- **Velocity vector:** Movement direction and speed

Coordinate System

Important: Obstacle data uses XZ plane representation:

- **X axis:** Left/right relative to camera
- **Z axis:** Forward/backward from camera
- **No Y component:** Height information not included in obstacle data

Development Notes

Velocity Vectors: Currently published as zeros. Future Hammerhead updates will provide non-zero velocity values for dynamic obstacle tracking.

Troubleshooting

- **No data files:** Verify Hammerhead is running and publishing obstacle data
- **Empty files:** Check that obstacles are being detected in the camera view
- **Permission errors:** Ensure write permissions in current directory
- **Build errors:** Ensure `hammerhead_msgs` package is built first

File Format Details

Each output file contains:

1. **Header section:** Format description and parameter order
2. **Data section:** Comma-separated values with obstacle information
3. **Timestamp:** File creation time in filename

2.2.4 Point Cloud Generator

Generate ROS2 `PointCloud2` messages from the `PointCloudSoup` messages published by Hammerhead.

Overview

This example demonstrates how to subscribe to bandwidth-efficient `PointCloudSoup` messages and convert them to standard `sensor_msgs/PointCloud2` messages for visualization and processing. The point clouds that Hammerhead generates can overwhelm the network due to bandwidth requirements. `PointCloudSoup` messages provide a compressed representation that can be losslessly reconstructed into XYZRGB point clouds while using a fraction of the bandwidth.

Build

```
cd hammerhead_ros2
colcon build --packages-up-to point_cloud_generator_py
```

Prerequisites

Configure Hammerhead to publish `PointCloudSoup` messages by modifying `master_config.ini` :

```
publish_point_cloud_type = 5
```

Usage

```
# Source the workspace
source install/setup.bash

# Run the point cloud generator
ros2 run point_cloud_generator_py point_cloud_generator_py
```

Features

- **Bandwidth Optimization:** Converts compressed `PointCloudSoup` messages to standard `PointCloud2`
- **Downsampling:** Reduces point cloud density by factor of 10 for network efficiency
- **Real-time Processing:** Publishes reconstructed point clouds in real-time
- **Quality of Service:** Uses `BEST_EFFORT` QoS policy for optimal performance

Topic Interface

SUBSCRIBED TOPICS

- `/nodar/point_cloud_soup` - Compressed point cloud data from Hammerhead

PUBLISHED TOPICS

- `/nodar/point_cloud` - Standard ROS2 point cloud messages (`sensor_msgs/PointCloud2`)

IMPORTANT RVIZ2 CONFIGURATION

This example publishes point clouds with `ReliabilityPolicy.BEST_EFFORT` QoS policy. In rviz2:

1. Add a `PointCloud2` display
2. Set topic to `/nodar/point_cloud`
3. Change **Reliability Policy** to **Best Effort**

Performance Tuning

ROS2 DDS CONFIGURATION

If you are having networking issues, please refer to the [ROS2 DDS tuning guide](#). For example, you may want to modify the fragmentation settings:

```
# Adjust IP fragmentation settings
sudo sysctl net.ipv4.ipfrag_time=3
sudo sysctl net.ipv4.ipfrag_high_thresh=536870912
```

PRODUCTION CONSIDERATIONS

- **Bandwidth:** Monitor network usage when publishing PointCloud2 messages
- **Downsampling:** Current example downsamples by factor of 10 - adjust as needed
- **QoS Settings:** Consider appropriate QoS policies for your application

Troubleshooting

- **No point clouds visible:** Verify Hammerhead is publishing PointCloudSoup messages
- **Poor performance:** Check network configuration and DDS settings
- **rviz2 display issues:** Ensure Reliability Policy is set to Best Effort
- **Memory issues:** Consider using C++ version for high-performance applications

2.2.5 Camera Parameter Control

Real-time camera parameter adjustment for Hammerhead via ROS2 services.

Overview

This example demonstrates how to control camera gain and exposure in real-time using ROS2 services. When ROS2 interfaces are enabled in Hammerhead's configuration, camera services become available for dynamic parameter adjustment during operation.

Build

```
cd hammerhead_ros2
colcon build --packages-up-to set_camera_params_py
```

Prerequisites

Ensure ROS2 interfaces are enabled in Hammerhead's `master_config.ini`.

Usage

```
# Source the workspace
source install/setup.bash

# Control camera exposure
ros2 run set_camera_params_py exposure

# Control camera gain
ros2 run set_camera_params_py gain
```

Features

- **Real-time Adjustment:** Modify camera parameters while Hammerhead is running
- **Interactive Interface:** Command-line interface for parameter control
- **Dual Control:** Separate tools for exposure and gain adjustment
- **Service-based:** Uses ROS2 services for reliable parameter setting

Service Interface

- `/nodar/set_exposure` - Camera exposure control
- `/nodar/set_gain` - Camera gain control

Both services use `hammerhead_msgs/CameraParam.srv`

Troubleshooting

- **Service not available:** Verify ROS2 interfaces are enabled in Hammerhead configuration
- **No response:** Check that Hammerhead is running and accessible
- **Invalid parameters:** Ensure parameter values are within valid ranges
- **Build errors:** Ensure `hammerhead_msgs` package is built first

2.3 C++ Examples

2.3.1 Generate ROS2 Bag

Convert data recorded by Hammerhead into ROS2 bag format for analysis and replay.

Build

```
cd hammerhead_ros2
colcon build --packages-up-to generate_rosbag2
```

Available Tools

xyz - POINT CLOUD ONLY

Generates a bag containing only `sensor_msgs/PointCloud2` messages with points containing `x,y,z` and `r,g,b` attributes.

everything - COMPLETE DATASET

Generates a bag containing the aforementioned point clouds as well as:

- Raw left and right camera images
- Rectified left camera image

Prerequisites

You need data recorded by Hammerhead in the following format:

```
20230208-133746/
disparity/
  000000000.tiff
  000000001.tiff
  ...
details/
  000000000.csv
  000000001.csv
  ...
```

Usage

```
# Source the workspace
source install/setup.bash

# Generate point cloud only bag
ros2 run generate_rosbag2 xyz <recorded_data_directory>

# Generate complete dataset bag
ros2 run generate_rosbag2 everything <recorded_data_directory>
```

EXAMPLES

```
# Generate complete bag from recorded data
ros2 run generate_rosbag2 everything 20230208-133746

# Save bag to custom location
ros2 run generate_rosbag2 everything 20230208-133746 ~/Downloads/my_bag
```

Output

The tool generates a ROS2 bag in the specified directory:

```
20230208-133746/
bag/
  bag_0.db3
  metadata.yaml
disparity/
```



```
000000000.tiff
...
details/
000000000.csv
...
```

Generated Topics

POINT CLOUD TOPICS

- `/nodar/point_cloud` - Generated 3D point cloud data

IMAGE TOPICS (WITH `everything` MODE)

- `/nodar/left/image_raw` - Raw left camera images
- `/nodar/right/image_raw` - Raw right camera images
- `/nodar/left/image_rect` - Rectified left camera images

Features

- Converts disparity data to 3D point clouds using standard stereo reconstruction
- Preserves timestamps from the original recording
- Generates standard ROS2 message types
- Configurable output location

Troubleshooting

- **Memory issues:** Use `xyz` mode for large datasets to reduce memory usage
- **Invalid path:** Check that the recorded data directory exists and contains the expected structure
- **Build errors:** Ensure all dependencies are installed and workspace is sourced

2.3.2 Image Viewer

Real-time OpenCV viewer for stereo images, disparity maps, and depth data published by Hammerhead.

Build

```
cd hammerhead_ros2
colcon build --packages-up-to image_viewer
```

Usage

```
# Source the workspace
source install/setup.bash

ros2 run image_viewer_py image_viewer_py <image_topic>
```

PARAMETERS

- `image_topic` : ROS2 topic name that provides `sensor_msgs::msg::Image` messages

EXAMPLES

```
# Source the workspace
source install/setup.bash

# View raw left camera
ros2 run image_viewer image_viewer /nodar/left/image_raw

# View disparity map
ros2 run image_viewer image_viewer /nodar/disparity
```

Available Image Topics

Topic	Description
<code>/nodar/left/image_raw</code>	Raw left camera
<code>/nodar/right/image_raw</code>	Raw right camera
<code>/nodar/left/image_rect</code>	Rectified left image
<code>/nodar/right/image_rect</code>	Rectified right image
<code>/nodar/disparity</code>	Disparity map
<code>/nodar/color_blended_depth/image_raw</code>	Color-coded depth visualization

Features

- Support for all image topic types
- Real-time display with OpenCV

Alternative Usage

You can also build and run this example using standalone CMake:

```
mkdir build && cd build
cmake .. && make
./image_viewer /nodar/left/image_raw
```

Integration with ROS2 Tools

There is nothing special about the image topics published by Hammerhead. You can also view them with tools like `rviz2` and `rqt`.

Troubleshooting

- **No display appears:** Check that Hammerhead is running and publishing image topics
- **Topic not found:** Verify the topic name using `ros2 topic list`
- **Build errors:** Ensure all dependencies are installed and workspace is sourced

Press `Ctrl+C` to exit the viewer.

2.3.3 Obstacle Data Recorder

A recorder for obstacle data published by Hammerhead.

Overview

This example demonstrates how to subscribe to `hammerhead_msgs/ObstacleData` messages from Hammerhead and save the detection data in structured text files for offline analysis and processing. The obstacle data is represented in the XZ plane (bird's eye view), where each obstacle is defined by:

- **Bounding box:** Collection of 3D points defining the obstacle perimeter
- **Velocity vector:** Movement direction and speed (currently in development)

Build

```
cd hammerhead_ros2
colcon build --packages-up-to obstacle_data_recorder
```

Usage

```
# Source the workspace
source install/setup.bash

# Run the obstacle data recorder
ros2 run obstacle_data_recorder obstacle_data_recorder
```

Features

- **Real-time Recording:** Subscribes to live obstacle detection data
- **Batch Processing:** Handles multiple obstacles per detection frame
- **Timestamped Data:** Preserves timing information for analysis

Topic Interface

SUBSCRIBED TOPICS

- `/nodar/obstacle_data` - Obstacle detection messages from Hammerhead

Output Format

The recorder creates an `obstacle_data` folder containing timestamped text files:

```
obstacle_data/
  obstacle_YYYYMMDD_HHMMSS_001.txt
  obstacle_YYYYMMDD_HHMMSS_002.txt
  ...
```

Each file contains:

- **Header:** Parameter descriptions and data format
- **Data:** Obstacle bounding box points and velocity vectors

DATA STRUCTURE

Each obstacle includes:

- **Bounding box points:** 3D coordinates (X, Z plane)
- **Velocity vector:** Movement direction and speed

Coordinate System

Important: Obstacle data uses XZ plane representation:

- **X axis:** Left/right relative to camera
- **Z axis:** Forward/backward from camera
- **No Y component:** Height information not included in obstacle data

Development Notes

Velocity Vectors: Currently published as zeros. Future Hammerhead updates will provide non-zero velocity values for dynamic obstacle tracking.

Troubleshooting

- **No data files:** Verify Hammerhead is running and publishing obstacle data
- **Empty files:** Check that obstacles are being detected in the camera view
- **Permission errors:** Ensure write permissions in current directory
- **Build errors:** Ensure `hammerhead_msgs` package is built first

File Format Details

Each output file contains:

1. **Header section:** Format description and parameter order
2. **Data section:** Comma-separated values with obstacle information
3. **Timestamp:** File creation time in filename

2.3.4 Point Cloud Generator

Generate ROS2 `PointCloud2` messages from the `PointCloudSoup` messages published by Hammerhead.

Overview

This example demonstrates how to subscribe to bandwidth-efficient `PointCloudSoup` messages and convert them to standard `sensor_msgs/PointCloud2` messages for visualization and processing. The point clouds that Hammerhead generates can overwhelm the network due to bandwidth requirements. `PointCloudSoup` messages provide a compressed representation that can be losslessly reconstructed into XYZRGB point clouds while using a fraction of the bandwidth.

Build

```
cd hammerhead_ros2
colcon build --packages-up-to point_cloud_generator
```

Prerequisites

Configure Hammerhead to publish `PointCloudSoup` messages by modifying `master_config.ini` :

```
publish_point_cloud_type = 5
```

Usage

```
# Source the workspace
source install/setup.bash

# Run the point cloud generator
ros2 run point_cloud_generator point_cloud_generator
```

Features

- **Bandwidth Optimization:** Converts compressed `PointCloudSoup` messages to standard `PointCloud2`
- **Downsampling:** Reduces point cloud density by factor of 10 for network efficiency
- **Real-time Processing:** Publishes reconstructed point clouds in real-time
- **Quality of Service:** Uses `BEST_EFFORT` QoS policy for optimal performance

Topic Interface

SUBSCRIBED TOPICS

- `/nodar/point_cloud_soup` - Compressed point cloud data from Hammerhead

PUBLISHED TOPICS

- `/nodar/point_cloud` - Standard ROS2 point cloud messages (`sensor_msgs/PointCloud2`)

IMPORTANT RVIZ2 CONFIGURATION

This example publishes point clouds with `ReliabilityPolicy.BEST_EFFORT` QoS policy. In rviz2:

1. Add a `PointCloud2` display
2. Set topic to `/nodar/point_cloud`
3. Change **Reliability Policy** to **Best Effort**

Performance Tuning

ROS2 DDS CONFIGURATION

If you are having networking issues, please refer to the [ROS2 DDS tuning guide](#). For example, you may want to modify the fragmentation settings:

```
# Adjust IP fragmentation settings
sudo sysctl net.ipv4.ipfrag_time=3
sudo sysctl net.ipv4.ipfrag_high_thresh=536870912
```

PRODUCTION CONSIDERATIONS

- **Bandwidth:** Monitor network usage when publishing PointCloud2 messages
- **Downsampling:** Current example downsamples by factor of 10 - adjust as needed
- **QoS Settings:** Consider appropriate QoS policies for your application

Troubleshooting

- **No point clouds visible:** Verify Hammerhead is publishing PointCloudSoup messages
- **Poor performance:** Check network configuration and DDS settings
- **rviz2 display issues:** Ensure Reliability Policy is set to Best Effort
- **Build errors:** Ensure hammerhead_msgs package is built first

2.3.5 Camera Parameter Control

Real-time camera parameter adjustment for Hammerhead via ROS2 services.

Overview

This example demonstrates how to control camera gain and exposure in real-time using ROS2 services. When ROS2 interfaces are enabled in Hammerhead's configuration, camera services become available for dynamic parameter adjustment during operation.

Build

```
cd hammerhead_ros2
colcon build --packages-up-to set_camera_params
```

Prerequisites

Ensure ROS2 interfaces are enabled in Hammerhead's `master_config.ini`.

Usage

```
# Source the workspace
source install/setup.bash

# Control camera exposure
ros2 run set_camera_params exposure

# Control camera gain
ros2 run set_camera_params gain
```

Features

- **Real-time Adjustment:** Modify camera parameters while Hammerhead is running
- **Interactive Interface:** Command-line interface for parameter control
- **Dual Control:** Separate tools for exposure and gain adjustment
- **Service-based:** Uses ROS2 services for reliable parameter setting

Service Interface

- `/nodar/set_exposure` - Camera exposure control
- `/nodar/set_gain` - Camera gain control

Both services use `hammerhead_msgs/CameraParam.srv`

Troubleshooting

- **Service not available:** Verify ROS2 interfaces are enabled in Hammerhead configuration
- **No response:** Check that Hammerhead is running and accessible
- **Invalid parameters:** Ensure parameter values are within valid ranges
- **Build errors:** Ensure `hammerhead_msgs` package is built first

3. ZMQ

3.1 Hammerhead ZMQ

A comprehensive Python and C++ client library for interfacing with the Hammerhead stereo vision system via ZeroMQ



3.1.1 Table of Contents




- [Overview](#)
- [Quick Start](#)
- [Message Types & Ports](#)
- [Project Structure](#)
- [Examples & Tutorials](#)
- [3D Coordinate System & Point Cloud Conversion](#)
- [API Reference](#)
- [Best Practices & Tips](#)

3.1.2 Overview

The Hammerhead system is a high-performance stereo vision processing unit that publishes various types of data over ZeroMQ. This library provides easy-to-use APIs for receiving and processing:

- **Stereo Images** - Raw and rectified left/right camera feeds
- **Depth Data** - Disparity maps and color-blended depth images
- **Point Clouds** - 3D point cloud data with or without RGB information
- **Obstacle Detection** - Real-time obstacle data with bounding boxes
- **Camera Control** - Parameter adjustment and scheduling

Key Features

Feature	Description
 Python Ready	Complete Python package with examples and utilities
 High Performance C++	Optimized C++ implementation for real-time applications
 ZeroMQ Protocol	Efficient, low-latency messaging with automatic reconnection

3.1.3 Quick Start

Repository Setup

```
# Get the Hammerhead ZMQ repository
git clone git@github.com:nodarhub/hammerhead_zmq.git

# If you received the HDK with version X.X.X, you can check out the corresponding tag (skip this step if you want the latest version):
git checkout X.X.X
```



```
# Make sure that the submodules are up to date
git submodule update --init --recursive
```

Python Installation & Usage

```
# Make a virtual environment (here we use ~/venvs/nodarenv, but you can choose any location)
mkdir -p ~/venvs && cd ~/venvs
python3 -m venv nodarenv

# Source the environment and install the package (from the root of the repository)
source ~/venvs/nodarenv/bin/activate
pip install -e .

# View live left raw image from Hammerhead device at 10.10.1.10
python examples/python/image_viewer/image_viewer.py 10.10.1.10 nodar/left/image_raw

# Record point clouds to PLY files from Hammerhead device at 10.10.1.10
python examples/python/point_cloud_recorder/point_cloud_recorder.py 10.10.1.10

# Record obstacle detection data from Hammerhead device at 10.10.1.10
python examples/python/obstacle_data_recorder/obstacle_data_recorder.py 10.10.1.10
```

C++ Installation & Usage

INSTALLING DEPENDENCIES

Ubuntu:

```
# Install build tools
sudo apt install build-essential cmake

# Install OpenCV (optional but recommended)
sudo apt install libopencv-dev

# Install libtiff (required for image_recorder)
sudo apt install libtiff-dev
```

Windows:

- **Visual Studio Community** - [Download](#)
- **CMake 3.11+** - [Download](#)
- **OpenCV 4.6.0+** - [Download](#)

BUILDING THE EXAMPLES & USAGE

```
# Build all examples
mkdir build && cd build
cmake .. && cmake --build . --config Release

# View live left raw image from Hammerhead device at 10.10.1.10
./examples/cpp/image_viewer/image_viewer 10.10.1.10 nodar/left/image_raw

# Record point clouds to PLY files from Hammerhead device at 10.10.1.10
./examples/cpp/point_cloud_recorder/point_cloud_recorder 10.10.1.10

# Record obstacle detection data from Hammerhead device at 10.10.1.10
./examples/cpp/obstacle_data_recorder/obstacle_data_recorder 10.10.1.10
```

3.1.4 Message Types & Ports

Hammerhead publishes data using structured message types over predefined ZMQ ports:

Image Streams

Port	Topic	Description	Message Type
9800	<code>nodar/left/image_raw</code>	Raw left camera feed	<code>StampedImage</code>
9801	<code>nodar/right/image_raw</code>	Raw right camera feed	<code>StampedImage</code>
9802	<code>nodar/left/image_rect</code>	Rectified left image	<code>StampedImage</code>
9803	<code>nodar/right/image_rect</code>	Rectified right image	<code>StampedImage</code>
9804	<code>nodar/disparity</code>	Disparity map (Q12.4 format)	<code>StampedImage</code>
9805	<code>nodar/color_blended_depth/image_raw</code>	Color-coded depth visualization	<code>StampedImage</code>
9813	<code>nodar/topbot_raw</code>	Raw top (left) and bottom (right) camera pair	<code>StampedImage</code>
9823	<code>nodar/topbot_rect</code>	Rectified top (left) and bottom (right) camera pair	<code>StampedImage</code>
9900	<code>nodar/occupancy_map</code>	Occupancy map	<code>StampedImage</code>

3D Data Streams

Port	Topic	Description	Message Type
9806	<code>nodar/point_cloud_soup</code>	Compact point cloud representation	<code>PointCloudSoup</code>
9809	<code>nodar/point_cloud</code>	Ordered point cloud	<code>PointCloud</code>
9810	<code>nodar/point_cloud_rgb</code>	RGB point cloud	<code>PointCloudRGB</code>

Detection & Control

Port	Topic	Description	Message Type
9807	<code>nodar/set_exposure</code>	Camera exposure control	<code>Control</code>
9808	<code>nodar/set_gain</code>	Camera gain control	<code>Control</code>
9811	<code>nodar/recording</code>	Recording on/off control	<code>SetBool</code>
9812	<code>nodar/obstacle</code>	Obstacle detection data	<code>ObstacleData</code>
9814	<code>nodar/wait</code>	Scheduler control	<code>SetBool</code>

3.1.5 Project Structure

The `zmq_msgs` folder contains the code for the `zmq_msgs` target, which defines how objects are sent and received via ZeroMQ on the network. It also defines other important networking information, such as which ports are used for which topics.

The `examples` folder contains comprehensive examples that demonstrate how to interact with Hammerhead. We envision that you will use these examples as a jumping-off point for your application.

We suggest that you start by examining the code and README's in the individual example directories for more details about what each example does.

3.1.6 Examples & Tutorials

Python Examples

Python examples provide easy-to-use scripts for common Hammerhead integration tasks.

VISUALIZATION EXAMPLES

- **Image Viewer** - Real-time OpenCV viewer for stereo images, disparity maps, and depth data

DATA CAPTURE EXAMPLES

- **Image Recorder** - Record images from any Hammerhead stream to disk as TIFF files
- **Point Cloud Recorder** - Subscribe to point cloud messages and save them as PLY files
- **Point Cloud Soup Recorder** - Stream the reduced-bandwidth PointCloudSoup messages, convert to point clouds, and save as PLY files
- **Obstacle Data Recorder** - Record real-time obstacle detection data

PROCESSING EXAMPLES

- **Depth to Disparity Converter** - Convert depth images to disparity format
- **Disparity to Point Cloud** - Convert stored disparity images to ordered 3D point clouds

CONTROL EXAMPLES

- **Hammerhead Scheduler** - Control Hammerhead's processing schedule
- **Topbot Publisher** - Publish images to Hammerhead's top/bottom camera topic

⚡ C++ Examples

High-performance C++ implementations for real-time applications and system integration.

VISUALIZATION EXAMPLES

- **Image Viewer** - Real-time OpenCV viewer for stereo images, disparity maps, and depth data

DATA CAPTURE EXAMPLES

- **Image Recorder** - Record images from any Hammerhead stream to disk as TIFF files
- **Point Cloud Recorder** - Subscribe to point cloud messages and save them as PLY files
- **Point Cloud Soup Recorder** - Stream the reduced-bandwidth PointCloudSoup messages, convert to point clouds, and save as PLY files
- **Obstacle Data Recorder** - Record real-time obstacle detection data

PROCESSING EXAMPLES

- **Offline Point Cloud Generator** - Batch processing of disparity images
- **Depth to Disparity Converter** - Convert depth images to disparity format
- **Legacy Obstacle Data Converter** - Convert legacy obstacle data formats

CONTROL EXAMPLES

- **Hammerhead Scheduler** - Control Hammerhead's processing schedule
- **Camera Parameter Control** - Real-time camera parameter adjustment
- **Topbot Publisher** - Publish images to Hammerhead's top/bottom camera topic

Common Integration Workflows

🧩 IMAGE PROCESSING PIPELINE

1. Start with **Image Viewer** to verify camera feeds
2. Use **Image Recorder** to capture datasets
3. Process images with custom algorithms

3D RECONSTRUCTION WORKFLOW

1. Subscribe to `PointCloudSoup` messages to reduce network bandwidth
2. Reconstruct point clouds
3. Process images with custom algorithms
4. Integrate with 3D processing frameworks

OBSTACLE DETECTION INTEGRATION

1. Use **Obstacle Data Recorder** to understand data format
2. Implement real-time processing of obstacle messages
3. Integrate with path planning or control systems
4. Add custom filtering or tracking algorithms

3.1.7 3D Coordinate System & Point Cloud Conversion

Hammerhead follows standard stereo reconstruction principles for converting disparity to 3D point clouds:

Disparity Scaling

The disparity is in Q12.4 format. We scale the disparity by `1 / 16.0` to get the disparity in `float32` format:

```
disparity_scaled = disparity.astype(np.float32) / 16.0
```

3D Reprojection

The scaled disparity map is reprojected into 3D space using OpenCV's `cv2.reprojectImageTo3D()` and a 4×4 reprojection matrix `Q`:

```
# Important: Negate the last row for correct coordinate frame
Q_corrected = Q.copy()
Q_corrected[3, :] = -Q_corrected[3, :]

# Reproject to 3D
points_3d = cv2.reprojectImageTo3D(disparity_scaled, Q_corrected)
```

A negative translation vector ($T_x < 0$) is used when creating the `Q` matrix to conform to the definition in OpenCV. This ensures that the point cloud is generated in a consistent right-handed coordinate frame. As a result, the entire last row of `Q` must be negated before passing to the `cv2.reprojectImageTo3D()` call.

This conversion scheme has been used in the following examples:

- **Offline Point Cloud Generator** - C++ batch processing
- **Point Cloud Soup Recorder** - C++ real-time recording
- **Point Cloud Soup Recorder** - Python real-time recording
- **Disparity to Point Cloud** - Python offline processing

3.1.8 API Reference

Message Types

All Hammerhead messages use a versioned protocol with the `MessageInfo` header structure.

STAMPEDIMAGE

Used for all image data including raw stereo images, rectified images, and disparity maps.

```
from zmq_msgs.image import StampedImage
import zmq

# Create ZMQ subscriber
context = zmq.Context()
socket = context.socket(zmq.SUB)
```



```

socket.connect(f"tcp://{ip_address}:{port}")
socket.setsockopt(zmq.SUBSCRIBE, b"")

# Receive and decode image
buffer = socket.recv()
stamped_image = StampedImage()
stamped_image.read(buffer)

# Access image data
cv2.imshow("Image", stamped_image.img)
print(f"Frame ID: {stamped_image.frame_id}")
print(f"Timestamp: {stamped_image.time}")

```

OBSTACLEDATA

Contains real-time obstacle detection information with bounding boxes and velocity vectors.

```

from zmq_msgs.obstacle_data import ObstacleData
import zmq

# Create ZMQ subscriber
context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect(f"tcp://{ip_address}:9812")
socket.setsockopt(zmq.SUBSCRIBE, b"")

# Receive and decode obstacle data
buffer = socket.recv()
obstacle_data = ObstacleData()
obstacle_data.read(buffer)

# Process obstacles
for obstacle in obstacle_data.obstacles:
    print(f"Bounding box: {obstacle.bounding_box.points}")
    print(f"Velocity: ({obstacle.velocity.x:.2f}, {obstacle.velocity.z:.2f}) m/s")

```

Coordinate System: Obstacle data is represented in the XZ plane (bird's eye view):

- **X axis:** Left/right relative to camera
- **Z axis:** Forward/backward from camera
- **No Y component:** Height information not included

3.1.9 Best Practices & Tips

Performance

- Use C++ for real-time applications
- Monitor network bandwidth with point clouds
- Implement connection timeouts
- Consider message buffering for high-frequency data

Reliability

- Always validate message types and versions
- Implement automatic reconnection logic
- Handle partial message reception
- Add logging for debugging network issues

Networking

- Test with different network conditions
- Use appropriate QoS settings if available
- Monitor for packet loss
- Consider compression for bandwidth-limited scenarios

Debugging

- Start with simple viewers before custom code
- Check Hammerhead configuration files
- Verify port availability and firewall settings
- Use network monitoring tools

3.2 Python Examples

3.2.1 Image Viewer

Real-time OpenCV viewer for stereo images, disparity maps, and depth data published by Hammerhead.

Installation

```
pip install -e examples/python/image_viewer
```

Usage

```
python image_viewer.py <src_ip> <image_topic_or_port>
```

PARAMETERS

- `src_ip`: IP address of the device running Hammerhead
- `image_topic_or_port`: Topic name or port number (see Available Topics below)

EXAMPLES

```
# View raw left camera
# Use 127.0.0.1 if running on the same device as Hammerhead
python image_viewer.py 127.0.0.1 nodar/left/image_raw

# Use the network IP address if running on a different device
python image_viewer.py 10.10.1.10 nodar/left/image_raw

# View disparity map
python image_viewer.py 10.10.1.10 9804

# View color-blended depth
python image_viewer.py 10.10.1.10 nodar/color_blended_depth/image_raw
```

Available Camera Topics

Topic	Port	Description
<code>nodar/left/image_raw</code>	9800	Raw left camera
<code>nodar/right/image_raw</code>	9801	Raw right camera
<code>nodar/left/image_rect</code>	9802	Rectified left image
<code>nodar/right/image_rect</code>	9803	Rectified right image
<code>nodar/disparity</code>	9804	Disparity map
<code>nodar/color_blended_depth/image_raw</code>	9805	Color-coded depth visualization
<code>nodar/topbot_raw</code>	9813	Raw top (left) and bottom (right) camera pair
<code>nodar/topbot_rect</code>	9823	Rectified top (left) and bottom (right) camera pair
<code>nodar/occupancy_map</code>	9900	Occupancy map

Features

- Support for all image topics (raw, rectified, disparity, depth)
- Real-time display with OpenCV

Troubleshooting

- **No display appears:** Check that Hammerhead is running and the IP address is correct
- **Connection timeout:** Verify network connectivity and firewall settings
- **Invalid topic:** Use one of the topics listed in the Available Topics section

Press `Ctrl+C` to exit the viewer.

3.2.2 Image Recorder

Record images from Hammerhead with ZMQ.

Installation

```
pip install -e examples/python/image_recorder
```

Usage

```
python image_recorder.py <src_ip> <image_topic_or_port> <output_dir>
```

PARAMETERS

- `src_ip`: IP address of the ZMQ source (the device running Hammerhead)
- `image_topic_or_port`: Topic name or port number for the image stream
- `output_dir`: Folder where the images will be saved

EXAMPLES

```
# Record raw right images using port number
python image_recorder.py 127.0.0.1 9801 raw_right_images

# Record raw right images using topic name
python image_recorder.py 127.0.0.1 nodar/right/image_raw output_dir
```

Available Camera Topics

Topic	Port	Description
<code>nodar/left/image_raw</code>	9800	Raw left camera
<code>nodar/right/image_raw</code>	9801	Raw right camera
<code>nodar/left/image_rect</code>	9802	Rectified left image
<code>nodar/right/image_rect</code>	9803	Rectified right image
<code>nodar/disparity</code>	9804	Disparity map
<code>nodar/color_blended_depth/image_raw</code>	9805	Color-coded depth visualization
<code>nodar/topbot_raw</code>	9813	Raw top (left) and bottom (right) camera pair
<code>nodar/topbot_rect</code>	9823	Rectified top (left) and bottom (right) camera pair

Output

The recorder creates a timestamped folder structure:

```
<output_dir>/YYYYMMDD-HHMMSS/
├── <topic_name>/          # Image subdirectory (e.g., left_raw/, topbot_raw/)
│   ├── 000000001.tiff
│   ├── 000000002.tiff
│   └── ...
├── times/                # Individual timestamp files
│   ├── 000000001.txt
│   ├── 000000002.txt
│   └── ...
```



```
|— times.txt      # Consolidated timestamps (one line per frame)
|— loop_latency.txt # Performance metrics
```

- **Images:** TIFF format with no compression
- **Naming:** 9-digit zero-padded frame numbers
- **Timestamps:** Each frame's timestamp(s) saved in `times/` and consolidated in `times.txt`
- **TOPBOT metadata:** For topbot images, dual timestamps (left/right) are embedded in TIFF metadata

Features

- Subscribe to any image topic published by Hammerhead
- Support for both topic names and port numbers
- Real-time recording with minimal latency

Troubleshooting

- **No images received:** Check IP address and ensure Hammerhead is running
- **Invalid topic:** Verify topic name exists in `topic_ports.py`
- **Connection hanging:** ZMQ will wait indefinitely for connection - check network connectivity

Press `Ctrl+C` to stop recording.

3.2.3 Point Cloud Recorder

Subscribe to point cloud messages and save them as PLY files for use in CloudCompare and other 3D software.

Installation

```
pip install -e examples/python/point_cloud_recorder
```

Usage

```
# Record standard point clouds
python point_cloud_recorder.py <src_ip>

# Record RGB point clouds (higher bandwidth)
python point_cloud_rgb_recorder.py <src_ip>
```

PARAMETERS

- `src_ip`: IP address of the device running Hammerhead

EXAMPLES

```
# Record point clouds from Hammerhead device
# Use 127.0.0.1 if running on the same device as Hammerhead
python point_cloud_recorder.py 127.0.0.1

# Use the network IP address if running on a different device
python point_cloud_recorder.py 10.10.1.10

# Record RGB point clouds with color information
python point_cloud_rgb_recorder.py 10.10.1.10
```

Output

- **Format:** PLY files saved in `point_clouds/` directory
- **Naming:** Sequential numbering (e.g., `cloud_000001.ply`)
- **Compatible with:** CloudCompare and other 3D visualization software

Features

- Saves point clouds as PLY files
- Support for both standard and RGB point clouds
- Compatible with CloudCompare and other 3D visualization software
- Progress tracking with timestamps

Bandwidth Warning

Important: Point cloud streaming requires significant network bandwidth. RGB point clouds require even more.

Troubleshooting

- **No files created:** Check that Hammerhead is running and point cloud streaming is enabled
- **Connection timeout:** Verify network connectivity and firewall settings
- **High bandwidth usage:** Consider using point cloud soup recorder for reduced bandwidth

Press `Ctrl+C` to stop recording.

3.2.4 Point Cloud Soup Recorder

Reconstruct high-resolution point clouds from Hammerhead's `PointCloudSoup` messages, and record as PLY files.

Installation

```
pip install -e examples/python/point_cloud_soup_recorder
```

Usage

```
python point_cloud_soup_recorder.py [hammerhead_ip] [output_directory]
```

PARAMETERS

- `hammerhead_ip`: IP address of the device running Hammerhead (default: 127.0.0.1)
- `output_directory`: Directory to save PLY files (default: `point_clouds` folder)

EXAMPLES

```
# Record point clouds from local device (default)
python point_cloud_soup_recorder.py

# Record point clouds from local device with custom output directory
python point_cloud_soup_recorder.py 127.0.0.1 /tmp/ply_output

# Record point clouds from remote device with custom output directory
python point_cloud_soup_recorder.py 10.10.1.10 /tmp/ply_output
```

Output

- **Format:** PLY files (`.ply`)
- **Location:** `point_clouds` folder (or specified directory)
- **Naming:** Sequential numbering based on frame IDs from messages

Features

- Subscribe to `PointCloudSoup` messages from Hammerhead
- Reconstruct full point clouds from compact soup representation
- Generate PLY files compatible with CloudCompare and other tools
- Handle high-resolution point clouds efficiently

`PointCloudSoup` Format

Nodar generates extremely high-resolution point clouds that require efficient network transmission. The `PointCloudSoup` format:

- Provides a compact representation of point cloud data
- Allows reconstruction of full point clouds on client machines
- Reduces network bandwidth requirements significantly
- Maintains high fidelity of 3D spatial information

Troubleshooting

- **No point clouds generated:** Check IP address and ensure Hammerhead is running
- **Connection hanging:** ZMQ will wait indefinitely for connection - verify network connectivity
- **Large file sizes:** Point clouds are high resolution - ensure adequate storage space

Press **Ctrl+C** to stop recording.

3.2.5 Obstacle Data Recorder

Record obstacle detection data from Hammerhead and save as text files.

Installation

```
pip install -e examples/python/obstacle_data_recorder
```

Usage

```
python obstacle_data_recorder.py <src_ip>
```

PARAMETERS

- **src_ip**: IP address of the ZMQ source (the device running Hammerhead)

EXAMPLES

```
# Record obstacle data from local device
python obstacle_data_recorder.py 127.0.0.1

# Record obstacle data from remote device
python obstacle_data_recorder.py 10.10.1.10
```

Output

- **Format**: Text files (.txt)
- **Location**: `obstacle_datas` folder
- **Naming**: Sequential numbering with timestamp information

Data Format

The obstacle data is represented in the XZ plane, where each obstacle is defined by:

- Bounding box coordinates
- Velocity vector (no vertical Y-axis component)

Each generated file contains:

- Header with parameter order information
- Obstacle bounding box data
- Velocity vector data

Features

- Subscribe to `ObstacleData` messages from Hammerhead
- Automatic text file generation with headers
- Real-time obstacle data recording
- XZ plane representation for 2D obstacle tracking

Troubleshooting

- **No data received**: Check IP address and ensure Hammerhead is running
- **Connection hanging**: ZMQ will wait indefinitely for connection - verify network connectivity
- **Empty files**: Ensure Hammerhead is actively detecting obstacles

Press **Ctrl+C** to stop recording.

3.2.6 Occupancy Map Viewer

Real-time viewer for occupancy grid maps with coordinate overlay, published by Hammerhead's GridDetect feature.

Installation

```
pip install -e examples/python/occupancy_map_viewer
```

Usage

```
python occupancy_map_viewer.py <src_ip>
```

PARAMETERS

- `src_ip`: IP address of the device running Hammerhead

EXAMPLES

```
# View occupancy map from local device
python occupancy_map_viewer.py 127.0.0.1

# View occupancy map from remote device
python occupancy_map_viewer.py 10.10.1.10
```

Features

- Binary occupancy grid visualization (white = occupied, black = free)
- Grid overlay with 10-meter spacing in physical coordinates
- Coordinate labels in margins (red for X-axis lateral, green for Z-axis depth)
- Real-time frame statistics: timestamp, grid dimensions, occupied cell count
- Metadata display: grid bounds (xMin, xMax, zMin, zMax) and cell size

Prerequisites

Important: Enable GridDetect in Hammerhead configuration:

- Set `enable_grid_detect = 1` in `master_config.ini`
- This automatically enables occupancy map publishing via ZMQ (port 9900)

Topic Information

Topic	Port	Description
<code>nodar/occupancy_map</code>	9900	Binary occupancy grid (CV_8UC1) with metadata

Troubleshooting

- **No display appears:** Check that `enable_grid_detect = 1` in `master_config.ini`
- **Connection timeout:** Verify network connectivity and that Hammerhead is running

Press `Ctrl+C` to exit the viewer.

3.2.7 Depth to Disparity Converter

Convert EXR depth images to TIFF disparity images for use with the Nodar Viewer.

Installation

```
pip install -e examples/python/depth_to_disparity
```

Usage

```
cd examples/python/depth_to_disparity/depth_to_disparity
OPENCV_IO_ENABLE_OPENEXR=1 python3 depth_to_disparity <data_directory> [output_directory]
```

PARAMETERS

- **data_directory** : Path to directory containing **depth** and **details** folders from Hammerhead
- **output_directory** : Optional output directory (defaults to **disparity** folder in data_directory)

EXAMPLES

```
# Convert depth images with default output location
OPENCV_IO_ENABLE_OPENEXR=1 python3 depth_to_disparity /path/to/hammerhead/data

# Convert depth images to specific output directory
OPENCV_IO_ENABLE_OPENEXR=1 python3 depth_to_disparity /path/to/hammerhead/data /path/to/output
```

Output

- **Format**: TIFF disparity images (.tiff)
- **Location**: **disparity** folder in data directory (or specified output directory)
- **Naming**: Maintains original depth image naming convention

Features

- Convert EXR depth images to lossless TIFF disparity format
- Compatible with Nodar Viewer for point cloud generation
- Preserves depth information accuracy
- Batch processing of entire directories

Requirements

- OpenCV with EXR support enabled
- Both **depth** and **details** folders in input directory
- Details data must be in YAML format

Troubleshooting

- **EXR support missing**: Install OpenCV with EXR support or use `OPENCV_IO_ENABLE_OPENEXR=1`
- **Missing details folder**: Ensure both **depth** and **details** folders exist in data directory
- **File overwrite warning**: Existing files in output directory will be overwritten
- **Orin compatibility**: Default OpenCV on Orin may lack EXR support - use x86-64 system

Press `Ctrl+C` to stop conversion.

3.2.8 Disparity to Ordered Point Cloud Converter

Convert TIFF disparity images to ordered point clouds.

Installation

```
pip install -e examples/python/disparity_to_ordered_point_cloud
```

Usage

```
cd examples/python/disparity_to_ordered_point_cloud/disparity_to_ordered_point_cloud
python3 disparity_to_ordered_point_cloud.py <disparity_dir> <details_dir> <output_dir> [--split]
```

PARAMETERS

- **disparity_dir**: Path to directory containing TIFF disparity images
- **details_dir**: Path to directory containing YAML details files
- **output_dir**: Directory where ordered point clouds will be saved
- **--split**: Optional flag to save XYZ dimensions as separate text files

EXAMPLES

```
# Generate 3-channel TIFF point clouds (default)
python3 disparity_to_ordered_point_cloud.py /path/to/disparity /path/to/details /path/to/output

# Generate separate text files for each XYZ dimension
python3 disparity_to_ordered_point_cloud.py /path/to/disparity /path/to/details /path/to/output --split
```

Output

- **Format**: 3-channel TIFF files (default) or separate text files (with --split)
- **Location**: Specified output directory
- **Naming**: Maintains original disparity image naming convention

Features

- Convert TIFF disparity images to ordered point clouds
- Support for both unified and split output formats
- Preserves spatial ordering of 3D points
- Compatible with various 3D visualization tools

Output Formats

DEFAULT (3-CHANNEL TIFF)

- Single TIFF file per disparity image
- Three channels representing X, Y, Z coordinates
- Maintains pixel-to-point correspondence

SPLIT FORMAT (--SPLIT OPTION)

- Three separate text files per disparity image
- Individual files for X, Y, Z dimensions
- Human-readable format for analysis

Requirements

- TIFF disparity images from depth_to_disparity converter
- YAML details files from Hammerhead data collection
- Sufficient storage space for point cloud data

Troubleshooting

- **Missing details files:** Ensure details directory contains corresponding YAML files
- **File overwrite warning:** Existing files in output directory will be overwritten
- **Large output files:** Point clouds can be very large - ensure adequate storage
- **Memory issues:** Processing large disparity images may require significant RAM

Press `Ctrl+C` to stop conversion.

3.2.9 Hammerhead Scheduler

Control Hammerhead's processing schedule to avoid resource conflicts with other intensive processes.

Installation

```
pip install -e examples/python/hammerhead_scheduler
```

Usage

```
python hammerhead_scheduler.py <src_ip>
```

PARAMETERS

- `src_ip`: IP address of the device running Hammerhead

EXAMPLES

```
# Control Hammerhead processing schedule
# Use 127.0.0.1 if running on the same device as Hammerhead
python hammerhead_scheduler.py 127.0.0.1

# Use the network IP address if running on a different device
python hammerhead_scheduler.py 10.10.1.10
```

Configuration Required

In Hammerhead's `master_config.ini` file, set:

```
wait_for_scheduler = 1
wait_for_scheduler_timeout_ms = 5000
```

How It Works

1. Hammerhead processes a frame
2. Sends scheduler request with frame number
3. Waits for scheduler reply before next frame
4. Times out after configured milliseconds if no reply

Features

- Microsecond-level timing control
- Prevents resource conflicts with other processes
- Configurable timeout protection
- Frame-by-frame processing control

Use Cases

- Coordinating with other intensive processes
- Custom frame rate control
- Synchronization with external systems
- Resource management on constrained systems

Troubleshooting

- **No scheduler activity:** Check that `wait_for_scheduler = 1` in `master_config.ini`

- **Timeout errors:** Adjust `wait_for_scheduler_timeout_ms` value
- **Connection issues:** Verify network connectivity and IP address

Press `Ctrl+C` to stop the scheduler.

3.2.10 Topbot Publisher

Publish vertically stacked stereo images (which we refer to as `topbot` images) from disk to Hammerhead.

Installation

```
pip install -e examples/python/topbot_publisher
```

Usage

```
python topbot_publisher <topbot_data_directory> <port_number> [pixel_format]
```

PARAMETERS

- `topbot_data_directory` : Path to directory containing sequentially numbered topbot images
- `port_number` : Port number to publish the images to
- `pixel_format` : Optional pixel format (default: BGR)

EXAMPLES

```
# Publish topbot images with default BGR format
python topbot_publisher /path/to/topbot/data 5000

# Publish topbot images with Bayer format
python topbot_publisher /path/to/topbot/data 5000 Bayer_RGGB
```

Supported Pixel Formats

- `BGR` (default)
- `Bayer_RGGB`
- `Bayer_GRBG`
- `Bayer_BGGR`
- `Bayer_GBRG`

Features

- Publish pre-recorded stereo image pairs to Hammerhead
- Single-pass playback of numbered image sequences (no looping)
- ZMQ-based communication for real-time streaming
- Multiple pixel format support

Requirements

- Images must be sequentially numbered
- Images must be in TIFF format
- Directory should contain only topbot image files
- Sufficient network bandwidth for real-time streaming

Troubleshooting

- **Images not found:** Ensure the directory contains sequentially numbered TIFF files
- **Connection issues:** Verify port number and network connectivity
- **File format errors:** Ensure all images are valid TIFF files

Press **Ctrl+C** to stop publishing.

3.2.11 QA Findings Viewer

Real-time viewer for quality assurance findings published by Hammerhead. Displays system monitoring, Hammerhead performance metrics, and image quality assessments.

Installation

```
pip install -e examples/python/qa_findings_viewer
```

Usage

```
python qa_findings_viewer.py <src_ip>
```

PARAMETERS

- `src_ip`: IP address of the device running Hammerhead

EXAMPLES

```
# View QA findings from local Hammerhead instance
python qa_findings_viewer.py 127.0.0.1

# View QA findings from remote Hammerhead device
python qa_findings_viewer.py 10.10.1.10
```

QA Findings Types

The viewer displays three categories of quality assurance findings:

SYSTEM MONITORING

- **CPU Temperature:** Warning $\geq 90^{\circ}\text{C}$, Error $\geq 99^{\circ}\text{C}$
- **GPU Temperature:** Warning $\geq 90^{\circ}\text{C}$, Error $\geq 99^{\circ}\text{C}$
- **GPU Utilization:** Warning $\geq 99.0\%$, Error $\geq 99.5\%$
- **GPU Memory:** Warning $\geq 80\%$, Error $\geq 90\%$
- **RAM Usage:** Warning $\geq 80\%$, Error $\geq 90\%$
- **CPU Load:** Warning ≥ 11.0 , Error ≥ 20.0 (load average)
- **Disk Space:** Warning $\leq 10\text{GB}$ free per mount point

HAMMERHEAD MONITORING

- **Frame Sync:** Warning $\geq 50\mu\text{s}$, Error $\geq 1000\mu\text{s}$ (synchronization timing between cameras)

IMAGE MONITORING

- **Underexposure:** Warning ≤ -1.5 (exposure assessment)
- *Detection method:* Analyzes pixel intensity histograms to identify when too many pixels are dark/clipped to black
- *Metric meaning:* Negative values indicate underexposure severity; more negative = more underexposed
- **Overexposure:** Warning ≥ 0.3 (exposure assessment)
- *Detection method:* Analyzes pixel intensity histograms to identify when too many pixels are bright/clipped to white
- *Metric meaning:* Positive values indicate overexposure severity; higher values = more overexposed
- **Blur:** Warning ≤ 0.58 (image sharpness, lower = more blurry)
- *Detection method:* Uses DFT (Discrete Fourier Transform) to analyze frequency domain energy content
- *Metric meaning:* Higher energy in high frequencies indicates sharper images; values below 0.58 indicate insufficient sharpness for reliable stereo matching

Output Format

Each QA finding is displayed with:

- **Frame ID:** Sequential frame identifier from the stereo camera system
- **Timestamp:** When the finding was detected (corresponds to left image timestamp)
- **Domain:** Category (system/hammerhead/image)
- **Key:** Specific metric identifier
- **Severity:** ERROR, WARNING, or INFO
- **Message:** Human-readable description
- **Value:** Numeric measurement
- **Unit:** Measurement unit

A summary is provided showing the total count of INFO, WARNING, and ERROR findings for each frame.

Monitoring Frequency

- **System Monitoring:** Periodic checks every 5000ms (5 seconds)
- **Hammerhead Monitoring:** Per-frame checks
- **Image Monitoring:** Per-frame checks

Configuration

To enable QA findings publishing in Hammerhead, set the following in `master_config.ini`:

```
# Enable quality assurance monitoring
enable_quality_assurance = 1

# Enable specific monitoring types
enable_system_monitor = 1
enable_hammerhead_monitor = 1
enable_image_monitor = 1

# Enable QA findings publishing
publish_qa_findings = 1
```

Features

- Real-time QA findings display with Python
- Frame ID tracking
- Persistent connection handling

Troubleshooting

GENERAL ISSUES

- **No findings appear:** Check that Hammerhead is running with QA enabled and `publish_qa_findings = 1`
- **Connection timeout:** Verify network connectivity and IP address

SYSTEM MONITORING ISSUES

- **High CPU/GPU Temperature:** Improve cooling, check thermal paste, verify fan operation
- **High RAM Memory:** Close unnecessary applications, increase swap space if needed
- **High CPU Load:** Identify resource-intensive processes, reduce concurrent operations
- **Low Disk Space:** Clean up temporary files, archive old data, expand storage

HAMMERHEAD MONITORING ISSUES

- **Frame Sync Errors:** Check camera connections, verify cable integrity, ensure cameras are properly synchronized

- **If you observe sparse or empty depth maps:**

- Perform initial calibration routine if mounting cameras for the first time
- Perform refinement calibration periodically
- Check for obstructions in camera field of view
- Ensure cameras are clean and focused
- Improve lighting conditions

IMAGE MONITORING ISSUES

- **Underexposure:** Increase lighting, adjust camera exposure settings, check for lens obstruction
- **Overexposure:** Reduce lighting intensity, adjust camera exposure settings
- **High Blur:** Clean camera lenses, check focus settings, reduce camera shake, verify mounting stability

Press **Ctrl+C** to exit the viewer.

3.3 C++ Examples

3.3.1 Image Viewer

Real-time OpenCV viewer for stereo images, disparity maps, and depth data published by Hammerhead.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./image_viewer <src_ip> <image_topic_or_port>

# Windows
./Release/image_viewer.exe <src_ip> <image_topic_or_port>
```

PARAMETERS

- `src_ip` : IP address of the device running Hammerhead
- `image_topic_or_port` : Topic name or port number (see Available Topics below)

EXAMPLES

```
# View raw left camera
# Use 127.0.0.1 if running on the same device as Hammerhead
./image_viewer 127.0.0.1 nodar/left/image_raw

# Use the network IP address if running on a different device
./image_viewer 10.10.1.10 nodar/left/image_raw

# View disparity map
./image_viewer 10.10.1.10 9804

# View color-blended depth
./image_viewer 10.10.1.10 nodar/color_blended_depth/image_raw
```

Available Camera Topics

Topic	Port	Description
<code>nodar/left/image_raw</code>	9800	Raw left camera
<code>nodar/right/image_raw</code>	9801	Raw right camera
<code>nodar/left/image_rect</code>	9802	Rectified left image
<code>nodar/right/image_rect</code>	9803	Rectified right image
<code>nodar/disparity</code>	9804	Disparity map
<code>nodar/color_blended_depth/image_raw</code>	9805	Color-coded depth visualization
<code>nodar/topbot_raw</code>	9813	Raw top (left) and bottom (right) camera pair
<code>nodar/topbot_rect</code>	9823	Rectified top (left) and bottom (right) camera pair
<code>nodar/occupancy_map</code>	9900	Occupancy map

Features

- Optimized for real-time performance

- Support for all image topic types

Troubleshooting

- **No display appears:** Check that Hammerhead is running and the IP address is correct
- **Connection timeout:** Verify network connectivity and firewall settings
- **Invalid topic:** Use one of the topics listed in the Available Topics section

Press `Ctrl+C` to exit the viewer.

3.3.2 Image Recorder

Record images from Hammerhead with ZMQ.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./image_recorder <src_ip> <image_topic_or_port> <output_dir>

# Windows
./Release/image_recorder.exe <src_ip> <image_topic_or_port> <output_dir>
```

PARAMETERS

- **src_ip**: IP address of the ZMQ source (the device running Hammerhead)
- **image_topic_or_port**: Topic name or port number for the image stream
- **output_dir**: Folder where the images will be saved

EXAMPLES

```
# Record raw right images using port number
./image_recorder 127.0.0.1 9801 raw_right_images

# Record raw right images using topic name
./image_recorder 127.0.0.1 nodar/right/image_raw raw_right_images
```

Available Camera Topics

Topic	Port	Description
nodar/left/image_raw	9800	Raw left camera
nodar/right/image_raw	9801	Raw right camera
nodar/left/image_rect	9802	Rectified left image
nodar/right/image_rect	9803	Rectified right image
nodar/disparity	9804	Disparity map
nodar/color_blended_depth/image_raw	9805	Color-coded depth visualization
nodar/topbot_raw	9813	Raw top (left) and bottom (right) camera pair
nodar/topbot_rect	9823	Rectified top (left) and bottom (right) camera pair

Output

The recorder creates a timestamped folder structure:

```
<output_dir>/YYYYMMDD-HHMMSS/
├── <topic_name>/          # Image subdirectory (e.g., left_raw/, topbot_raw/)
│   ├── 000000001.tiff
│   ├── 000000002.tiff
│   └── ...
├── times/                # Individual timestamp files
│   ├── 000000001.txt
│   └── 000000002.txt
```



```

└─ times.txt
# Consolidated timestamps (one line per frame)

```

- **Images:** TIFF format with no compression
- **Naming:** 9-digit zero-padded frame numbers
- **Timestamps:** Each frame's timestamp(s) saved in `times/` and consolidated in `times.txt`
- **TOPBOT metadata:** For topbot images, dual timestamps (left/right) are embedded in TIFF metadata

Features

- High-performance C++ implementation for minimal latency
- Subscribe to any image topic published by Hammerhead
- Support for both topic names and port numbers
- Real-time recording with optimized memory usage

Topic to Port Mapping

The file `topic_ports.hpp` defines the `topic->port` mappings. For example:

- `"nodar/right/image_raw"` maps to port `9801`

Troubleshooting

- **No images received:** Check IP address and ensure Hammerhead is running
- **Invalid topic:** Verify topic name exists in `topic_ports.hpp`
- **Connection hanging:** ZMQ will wait indefinitely for connection - check network connectivity

Press `Ctrl+C` to stop recording.

3.3.3 Point Cloud Recorder

Subscribe to point cloud messages and save them as PLY files for use in CloudCompare and other 3D software.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux - Record standard point clouds
./point_cloud_recorder <src_ip>

# Linux - Record RGB point clouds (higher bandwidth)
./point_cloud_rgb_recorder <src_ip>

# Windows - Record standard point clouds
./Release/point_cloud_recorder.exe <src_ip>

# Windows - Record RGB point clouds (higher bandwidth)
./Release/point_cloud_rgb_recorder.exe <src_ip>
```

PARAMETERS

- **src_ip**: IP address of the device running Hammerhead

EXAMPLES

```
# Record point clouds from Hammerhead device
# Use 127.0.0.1 if running on the same device as Hammerhead
./point_cloud_recorder 127.0.0.1

# Use the network IP address if running on a different device
./point_cloud_recorder 10.10.1.10

# Record RGB point clouds with color information
./point_cloud_rgb_recorder 10.10.1.10
```

Output

- **Format**: PLY files saved in `point_clouds/` directory
- **Naming**: Sequential numbering (e.g., `cloud_000001.ply`)
- **Compatible with**: CloudCompare and other 3D visualization software

Features

- Optimized memory allocation
- Fast PLY file writing
- Support for RGB point clouds
- Real-time performance monitoring
- Progress tracking with timestamps

Bandwidth Warning

Important: Point cloud streaming requires significant network bandwidth. RGB point clouds require even more.

Troubleshooting

- **No files created**: Check that Hammerhead is running and point cloud streaming is enabled

- **Connection timeout:** Verify network connectivity and firewall settings
- **High bandwidth usage:** Consider using point cloud soup recorder for reduced bandwidth

Press `Ctrl+C` to stop recording.

3.3.4 Point Cloud Soup Recorder

Reconstruct high-resolution point clouds from Hammerhead's `PointCloudSoup` messages, and record as PLY files.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./point_cloud_soup_recorder [OPTIONS] [hammerhead_ip] [output_directory]

# Windows
./Release/point_cloud_soup_recorder.exe [OPTIONS] [hammerhead_ip] [output_directory]
```

OPTIONS

- `-w`, `--wait-for-scheduler`: Enable scheduler synchronization with Hammerhead. When enabled, the recorder will wait for Hammerhead to request the next frame before continuing, ensuring frame-by-frame synchronization.
- `-h`, `--help`: Display usage information

PARAMETERS

- `hammerhead_ip`: IP address of the device running Hammerhead (default: 127.0.0.1)
- `output_directory`: Directory to save PLY files (default: `point_clouds` folder)

EXAMPLES

```
# Record point clouds from local device (default)
./point_cloud_soup_recorder

# Record point clouds from local device with custom output directory
./point_cloud_soup_recorder 127.0.0.1 /tmp/ply_output

# Record point clouds from remote device with custom output directory
./point_cloud_soup_recorder 10.10.1.10 /tmp/ply_output

# Record with scheduler synchronization (frame-by-frame)
./point_cloud_soup_recorder -w 10.10.1.10 /tmp/ply_output
./point_cloud_soup_recorder --wait-for-scheduler 10.10.1.10 /tmp/ply_output
```

SCHEDULER SYNCHRONIZATION WORKFLOW

When using the `-w` flag for frame-by-frame synchronization:

1. Configure Hammerhead - Edit `master_config.ini`:

```
wait_for_scheduler = 1
```

2. Start the recorder first:

```
./point_cloud_soup_recorder -w
```

3. Start Hammerhead - The recorder will now control frame processing timing

This ensures proper ZMQ connection establishment.

Output

- **Format:** PLY files (`.ply`)
- **Location:** `point_clouds` folder
- **Naming:** Sequential numbering based on received messages

Features

- High-performance C++ implementation for optimal processing speed
- Subscribe to `PointCloudSoup` messages from Hammerhead
- Reconstruct full point clouds from compact soup representation
- Generate PLY files compatible with CloudCompare and other tools
- Handle high-resolution point clouds efficiently with minimal memory usage
- Optional scheduler synchronization for frame-by-frame processing with Hammerhead

`PointCloudSoup` Format

Nodar generates extremely high-resolution point clouds that require efficient network transmission. The `PointCloudSoup` format:

- Provides a compact representation of point cloud data
- Allows reconstruction of full point clouds on client machines
- Reduces network bandwidth requirements significantly
- Maintains high fidelity of 3D spatial information

The `point_cloud_soup.hpp` header provides the class and functions for reading and writing `PointCloudSoup` data.

Troubleshooting

- **No point clouds generated:** Check IP address and ensure Hammerhead is running
- **Connection hanging:** ZMQ will wait indefinitely for connection - verify network connectivity
- **Large file sizes:** Point clouds are high resolution - ensure adequate storage space

Press `Ctrl+C` to stop recording.

3.3.5 Obstacle Data Recorder

Record obstacle detection data from Hammerhead and save as text files.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./obstacle_data_recorder <src_ip>

# Windows
./Release/obstacle_data_recorder.exe <src_ip>
```

PARAMETERS

- **src_ip**: IP address of the ZMQ source (the device running Hammerhead)

EXAMPLES

```
# Record obstacle data from local device
./obstacle_data_recorder 127.0.0.1

# Record obstacle data from remote device
./obstacle_data_recorder 10.10.1.10
```

Output

- **Format**: Text files (.txt)
- **Location**: `obstacle_data` folder
- **Naming**: Sequential numbering with timestamp information

Data Format

The obstacle data is represented in the XZ plane, where each obstacle is defined by:

- Bounding box coordinates
- Velocity vector (no vertical Y-axis component)

Each generated file contains:

- Header with parameter order information
- Obstacle bounding box data
- Velocity vector data

Features

- High-performance C++ implementation for real-time processing
- Subscribe to `ObstacleData` messages from Hammerhead
- Automatic text file generation with headers
- Real-time obstacle data recording
- XZ plane representation for 2D obstacle tracking

Troubleshooting

- **No data received:** Check IP address and ensure Hammerhead is running
- **Connection hanging:** ZMQ will wait indefinitely for connection - verify network connectivity
- **Empty files:** Ensure Hammerhead is actively detecting obstacles

Press `Ctrl+C` to stop recording.

3.3.6 Occupancy Map Viewer

Real-time viewer for occupancy grid maps published by Hammerhead's GridDetect feature.

This directory contains two versions:

- **occupancy_map_viewer** : Full version with OpenCV visualization (grid overlay, coordinate labels)
- **occupancy_map_stats** : Lightweight version without OpenCV (prints metadata and statistics only)

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Both executables will be built in the same directory.

Usage

OCCUPANCY_MAP_VIEWER (WITH VISUALIZATION)

```
# Linux
./occupancy_map_viewer <src_ip>

# Windows
./Release/occupancy_map_viewer.exe <src_ip>
```

Requires: OpenCV 4

OCCUPANCY_MAP_STATS (STATISTICS ONLY)

```
# Linux
./occupancy_map_stats <src_ip>

# Windows
./Release/occupancy_map_stats.exe <src_ip>
```

Requires: No OpenCV dependency - only ZMQ

PARAMETERS

- **src_ip** : IP address of the device running Hammerhead

EXAMPLES

```
# View occupancy map from local device
./occupancy_map_viewer 127.0.0.1

# Print occupancy statistics from remote device (no OpenCV needed)
./occupancy_map_stats 10.10.1.10
```

Features

OCCUPANCY_MAP_VIEWER

- Binary occupancy grid visualization (white = occupied, black = free)
- Grid overlay with 10-meter spacing in physical coordinates
- Coordinate labels in margins (red for X-axis lateral, green for Z-axis depth)
- Real-time frame statistics: timestamp, grid dimensions, occupied cell count
- Metadata display: grid bounds (xMin, xMax, zMin, zMax) and cell size

OCCUPANCY_MAP_STATS

- Prints frame statistics: frame ID, timestamp, grid dimensions, image type

- Displays occupied cell count and occupancy percentage
- Shows metadata: grid bounds (xMin, xMax, zMin, zMax), cell size, grid dimensions

Prerequisites

Important: Enable GridDetect in Hammerhead configuration:

- Set `enable_grid_detect = 1` in `master_config.ini`
- This automatically enables occupancy map publishing via ZMQ (port 9900)

Topic Information

Topic	Port	Description
<code>nodar/occupancy_map</code>	9900	Binary occupancy grid (CV_8UC1) with metadata

Troubleshooting

- **No display appears:** Check that `enable_grid_detect = 1` in `master_config.ini`
- **Connection timeout:** Verify network connectivity and that Hammerhead is running

Press `Ctrl+C` to exit the viewer.

3.3.7 Depth to Disparity Converter

Convert EXR or TIFF depth images to TIFF disparity images for use with the Nodar Viewer.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./depth_to_disparity <data_directory> [output_directory]

# Windows
./Release/depth_to_disparity.exe <data_directory> [output_directory]
```

PARAMETERS

- **data_directory**: Path to directory containing `depth` and `details` folders from Hammerhead
- **output_directory**: Optional output directory (defaults to `disparity` folder in `data_directory`)

EXAMPLES

```
# Convert depth images with default output location
./depth_to_disparity /path/to/hammerhead/data

# Convert depth images to specific output directory
./depth_to_disparity /path/to/hammerhead/data /path/to/output
```

Output

- **Format**: TIFF disparity images (.tiff)
- **Location**: `disparity` folder in data directory (or specified output directory)
- **Naming**: Maintains original depth image naming convention

Features

- High-performance C++ implementation for fast batch processing
- Convert EXR or TIFF depth images to lossless TIFF disparity format
- Compatible with Nodar Viewer for point cloud generation
- Preserves depth information accuracy
- Efficient memory usage for large image datasets

Requirements

- OpenCV with EXR support enabled
- Both `depth` and `details` folders in input directory
- Details data must be in YAML format

Troubleshooting

- **EXR support missing**: Install OpenCV with EXR support - available on most Ubuntu x86-64 installations
- **Missing details folder**: Ensure both `depth` and `details` folders exist in data directory
- **File overwrite warning**: Existing files in output directory will be overwritten

- **ARM compatibility:** Default OpenCV on ARM systems may lack EXR support - use x86-64 system

Press `Ctrl+C` to stop conversion.

3.3.8 Offline Point Cloud Generator

Convert saved Hammerhead data into point clouds and save as PLY files.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./offline_point_cloud_generator <data_directory> [output_directory]

# Windows
./Release/offline_point_cloud_generator.exe <data_directory> [output_directory]
```

PARAMETERS

- **data_directory**: Path to directory containing Hammerhead saved data
- **output_directory**: Optional output directory (defaults to `point_clouds` folder in `data_directory`)

EXAMPLES

```
# Generate point clouds with default output location
./offline_point_cloud_generator /path/to/hammerhead/data

# Generate point clouds to specific output directory
./offline_point_cloud_generator /path/to/hammerhead/data /path/to/output
```

Output

- **Format**: PLY files (.ply)
- **Location**: `point_clouds` folder in data directory (or specified output directory)
- **Naming**: Sequential numbering based on processed data

Features

- High-performance C++ implementation for fast batch processing
- Convert saved Hammerhead data into full point clouds
- Generate PLY files compatible with CloudCompare and other tools
- Efficient memory usage for large datasets
- Support for both EXR and TIFF depth formats

Requirements

- OpenCV with EXR support enabled (for legacy EXR files)
- Complete Hammerhead data directory with depth and calibration information

Troubleshooting

- **EXR support missing**: Install OpenCV with EXR support - available on most Ubuntu x86-64 installations
- **Missing data files**: Ensure data directory contains all required Hammerhead output files
- **ARM compatibility**: Default OpenCV on ARM systems may lack EXR support - use x86-64 system
- **Memory issues**: Large datasets may require significant RAM - monitor system resources

Press **Ctrl+C** to stop generation.

3.3.9 Hammerhead Scheduler

Control Hammerhead's processing schedule to avoid resource conflicts with other intensive processes.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./hammerhead_scheduler <src_ip>

# Windows
./Release/hammerhead_scheduler.exe <src_ip>
```

PARAMETERS

- `src_ip`: IP address of the device running Hammerhead

EXAMPLES

```
# Control Hammerhead processing schedule
# Use 127.0.0.1 if running on the same device as Hammerhead
./hammerhead_scheduler 127.0.0.1

# Use the network IP address if running on a different device
./hammerhead_scheduler 10.10.1.10
```

Configuration Required

In Hammerhead's `master_config.ini` file, set:

```
wait_for_scheduler = 1
wait_for_scheduler_timeout_ms = 5000
```

How It Works

1. Hammerhead processes a frame
2. Sends scheduler request with frame number
3. Waits for scheduler reply before next frame
4. Times out after configured milliseconds if no reply

Features

- Native C++ performance for low-latency scheduling
- Microsecond-level timing control
- Prevents resource conflicts with other processes
- Configurable timeout protection
- Frame-by-frame processing control

Troubleshooting

- **No scheduler activity:** Check that `wait_for_scheduler = 1` in `master_config.ini`
- **Timeout errors:** Adjust `wait_for_scheduler_timeout_ms` value
- **Connection issues:** Verify network connectivity and IP address

Press `Ctrl+C` to stop the scheduler.

3.3.10 Set Camera Parameters

Modify camera parameters in Hammerhead using ZMQ Request-Reply pattern.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./set_exposure <src_ip>
./set_gain <src_ip>

# Windows
./Release/set_exposure.exe <src_ip>
./Release/set_gain.exe <src_ip>
```

PARAMETERS

- **src_ip**: IP address of the ZMQ source (the device running Hammerhead)

EXAMPLES

```
# Set camera exposure
# Use 127.0.0.1 if running on the same device as Hammerhead
./set_exposure 127.0.0.1

# Use the network IP address if running on a different device
./set_exposure 10.10.1.10

# Set camera gain
./set_gain 10.10.1.10
```

Output

This example provides interactive control for: - Camera exposure settings - Camera gain settings

Features

- High-performance C++ implementation for real-time parameter control
- Request-Reply pattern for reliable parameter setting
- Interactive exposure control
- Interactive gain control
- Real-time parameter adjustment

Available Controls

- **Exposure**: Adjust camera exposure settings
- **Gain**: Modify camera gain parameters

The topic names and ports used for this communication are defined in the `topic_ports.hpp` header.

Troubleshooting

- **No response**: Check IP address and ensure Hammerhead is running
- **Connection hanging**: ZMQ will wait indefinitely for connection - verify network connectivity
- **Parameter not applied**: Ensure Hammerhead is configured to accept parameter changes

Press **Ctrl+C** to stop parameter control.

3.3.11 Topbot Publisher

Publish vertically stacked stereo images (which we refer to as `topbot` images) from disk to Hammerhead.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./topbot_publisher <topbot_data_directory> <port_number> [pixel_format]

# Windows
./Release/topbot_publisher.exe <topbot_data_directory> <port_number> [pixel_format]
```

PARAMETERS

- `topbot_data_directory` : Path to directory containing sequentially numbered topbot images
- `port_number` : Port number to publish the images to
- `pixel_format` : Optional pixel format (default: BGR)

EXAMPLES

```
# Publish topbot images with default BGR format
./topbot_publisher /path/to/topbot/data 5000

# Publish topbot images with Bayer format
./topbot_publisher /path/to/topbot/data 5000 Bayer_RGGB
```

Features

- High-performance C++ implementation for real-time streaming
- Publish pre-recorded stereo image pairs to Hammerhead
- Single-pass playback of numbered image sequences (no looping)
- ZMQ-based communication for minimal latency
- Multiple pixel format support

Supported Pixel Formats

- `BGR` (default)
- `Bayer_RGGB`
- `Bayer_GRGB`
- `Bayer_BGGR`
- `Bayer_GBRG`

Requirements

- Images must be sequentially numbered
- Images must be in TIFF format
- Directory should contain only topbot image files
- Sufficient network bandwidth for real-time streaming

Troubleshooting

- **Images not found:** Ensure the directory contains sequentially numbered TIFF files
- **Connection issues:** Verify port number and network connectivity
- **File format errors:** Ensure all images are valid TIFF files

Press `Ctrl+C` to stop publishing.

3.3.12 Legacy Obstacle Data Converter

Convert legacy `bounding_boxes_and_velocities` data to `tracked-objects` format for use with the Nodar Viewer.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./legacy_obstacle_data_converter <data_directory> [output_directory]

# Windows
./Release/legacy_obstacle_data_converter.exe <data_directory> [output_directory]
```

PARAMETERS

- **data_directory**: Path to directory containing `bounding_boxes_and_velocities` folder from Hammerhead
- **output_directory**: Optional output directory (defaults to `tracked-objects` folder in `data_directory`)

EXAMPLES

```
# Convert legacy data with default output location
./legacy_obstacle_data_converter /path/to/hammerhead/data

# Convert legacy data to specific output directory
./legacy_obstacle_data_converter /path/to/hammerhead/data /path/to/output
```

Output

- **Format**: Tracked-objects data files
- **Location**: `tracked-objects` folder in data directory (or specified output directory)
- **Naming**: Maintains original data naming convention

Features

- High-performance C++ implementation for fast batch conversion
- Convert legacy `bounding_boxes_and_velocities` data to modern `tracked-objects` format
- Compatible with Nodar Viewer for displaying object bounding boxes
- Preserves object tracking information and velocity data
- Efficient processing of large datasets

Use Case

This converter is needed when working with older Hammerhead data that used the legacy `bounding_boxes_and_velocities` format. The converted `tracked-objects` data can be displayed in the Nodar Viewer for visualization of detected objects with their bounding boxes and tracking information.

Requirements

- Data directory must contain `bounding_boxes_and_velocities` folder
- Legacy data must be in the expected format from older Hammerhead versions

Troubleshooting

- **Missing legacy folder:** Ensure `bounding_boxes_and_velocities` folder exists in data directory
- **File overwrite warning:** Existing files in output directory will be overwritten
- **Format errors:** Ensure legacy data is in the expected format from older Hammerhead versions

Press `Ctrl+C` to stop conversion.

3.3.13 QA Findings Viewer

Real-time viewer for quality assurance findings published by Hammerhead. Displays system monitoring, Hammerhead performance metrics, and image quality assessments.

Build

```
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Usage

```
# Linux
./qa_findings_viewer <src_ip>

# Windows
./Release/qa_findings_viewer.exe <src_ip>
```

PARAMETERS

- **src_ip**: IP address of the device running Hammerhead

EXAMPLES

```
# View QA findings from local Hammerhead instance
./qa_findings_viewer 127.0.0.1

# View QA findings from remote Hammerhead device
./qa_findings_viewer 10.10.1.10
```

QA Findings Types

The viewer displays three categories of quality assurance findings:

SYSTEM MONITORING

- **CPU Temperature**: Warning $\geq 90^{\circ}\text{C}$, Error $\geq 99^{\circ}\text{C}$
- **GPU Temperature**: Warning $\geq 90^{\circ}\text{C}$, Error $\geq 99^{\circ}\text{C}$
- **RAM Usage**: Warning $\geq 80\%$, Error $\geq 90\%$
- **CPU Load**: Warning ≥ 11.0 , Error ≥ 20.0 (load average)
- **Disk Space**: Warning $\leq 10\text{GB}$ free per mount point

HAMMERHEAD MONITORING

- **Frame Sync**: Warning $\geq 50\mu\text{s}$, Error $\geq 1000\mu\text{s}$ (synchronization timing between cameras)

IMAGE MONITORING

- **Underexposure:** Warning ≤ -1.5 (exposure assessment)
- *Detection method:* Analyzes pixel intensity histograms to identify when too many pixels are dark/clipped to black
- *Metric meaning:* Negative values indicate underexposure severity; more negative = more underexposed
- **Overexposure:** Warning ≥ 0.3 (exposure assessment)
- *Detection method:* Analyzes pixel intensity histograms to identify when too many pixels are bright/clipped to white
- *Metric meaning:* Positive values indicate overexposure severity; higher values = more overexposed
- **Blur:** Warning ≤ 0.58 (image sharpness, lower = more blurry)
- *Detection method:* Uses DFT (Discrete Fourier Transform) to analyze frequency domain energy content
- *Metric meaning:* Higher energy in high frequencies indicates sharper images; values below 0.58 indicate insufficient sharpness for reliable stereo matching

Output Format

Each QA finding is displayed with:

- **Frame ID:** Sequential frame identifier from the stereo camera system
- **Timestamp:** When the finding was detected (corresponds to left image timestamp)
- **Domain:** Category (system/hammerhead/image)
- **Key:** Specific metric identifier
- **Severity:** ERROR, WARNING, or INFO
- **Message:** Human-readable description
- **Value:** Numeric measurement
- **Unit:** Measurement unit

A summary is provided showing the total count of INFO, WARNING, and ERROR findings for each frame.

Monitoring Frequency

- **System Monitoring:** Periodic checks every 5000ms (5 seconds)
- **Hammerhead Monitoring:** Per-frame checks
- **Image Monitoring:** Per-frame checks

Configuration

To enable QA findings publishing in Hammerhead, set the following in `master_config.ini`:

```
# Enable quality assurance monitoring
enable_quality_assurance = 1

# Enable specific monitoring types
enable_system_monitor = 1
enable_hammerhead_monitor = 1
enable_image_monitor = 1

# Enable QA findings publishing
publish_qa_findings = 1
```

Features

- Real-time QA findings display
- Frame ID tracking
- Persistent connection handling

Troubleshooting

GENERAL ISSUES

- **No findings appear:** Check that Hammerhead is running with QA enabled and `publish_qa_findings = 1`
- **Connection timeout:** Verify network connectivity and IP address

SYSTEM MONITORING ISSUES

- **High CPU/GPU Temperature:** Improve cooling, check thermal paste, verify fan operation
- **High RAM Memory:** Close unnecessary applications, increase swap space if needed
- **High CPU Load:** Identify resource-intensive processes, reduce concurrent operations
- **Low Disk Space:** Clean up temporary files, archive old data, expand storage

HAMMERHEAD MONITORING ISSUES

- **Frame Sync Errors:** Check camera connections, verify cable integrity, ensure cameras are properly synchronized
- **If you observe sparse or empty depth maps:**
 - Perform initial calibration routine if mounting cameras for the first time
 - Perform refinement calibration periodically
 - Check for obstructions in camera field of view
 - Ensure cameras are clean and focused
 - Improve lighting conditions

IMAGE MONITORING ISSUES

- **Underexposure:** Increase lighting, adjust camera exposure settings, check for lens obstruction
- **Overexposure:** Reduce lighting intensity, adjust camera exposure settings
- **High Blur:** Clean camera lenses, check focus settings, reduce camera shake, verify mounting stability

Press `Ctrl+C` to exit the viewer.

4. Frequently Asked Questions

How can I generate point clouds from the recorded data?

To ensure real-time performance during data recording, point clouds are not saved directly. However, you can generate point clouds from the recorded data afterward. Follow this example to create point clouds offline:


[Example Code](#)
[Example Docs](#)

How can I generate ROS2 bags from the recorded data?

ROS2 bags are not directly recorded to ensure optimal performance during data capture. However, you can generate ROS2 bags from the recorded data afterward. Follow this example to generate ROS2 bags from recorded data:


[Example Code](#)
[Example Docs](#)

The 3D bounding boxes in `nodar_viewer` are not aligned with the ground

The 3D bounding boxes in `nodar_viewer` are often not aligned with the ground and/or have an offset with respect to the ground when the camera axis is not parallel with the ground plane. Most issues are fixed by adjusting `bar_pitch` in the `master_config.ini`.

In `nodar_viewer`, you can adjust this by going to

Viewers Tab -> Point Cloud -> -> Bounding Boxes -> Display Boxes ☒

Then tune the `Box Offset` parameter to adjust the relative position of the 3D bounding boxes along the `y` axis.

The distance to an object gives me the wrong value

During shipment and large shock events, the absolute range calibration can sometimes be affected. To recalibrate the range measurements, run `nodar_viewer`, and go to

Viewers Tab -> Initial Calibration -> -> Range Calibration -> Add Region

Then select a target in the image with a known distance, enter the distance to the target, and click `Submit`.

When the calibration is complete, recheck the distance to the target in the `Colorized Depth Map` (click a point in the image so see the distance). If the ranges still seem to be wrong, you should try to select a different target and repeat the range calibration routine. An ideal target would be relatively large and far away, that is, more than `30 m` or `100 ft` away.

The detection is cut off in the bottom left corner of the Colorized Depth Map

Shadows around objects and missing returns in the bottom left corner are expected. The bottom left corner having no returns is often due to the field of view. Hammerhead can only return depth estimates in regions where the fields-of-view of the left and right cameras overlap. The bottom left corner of the left image is often cut off because that portion of the image isn't visible in the right image.