Nodar 2.15.1

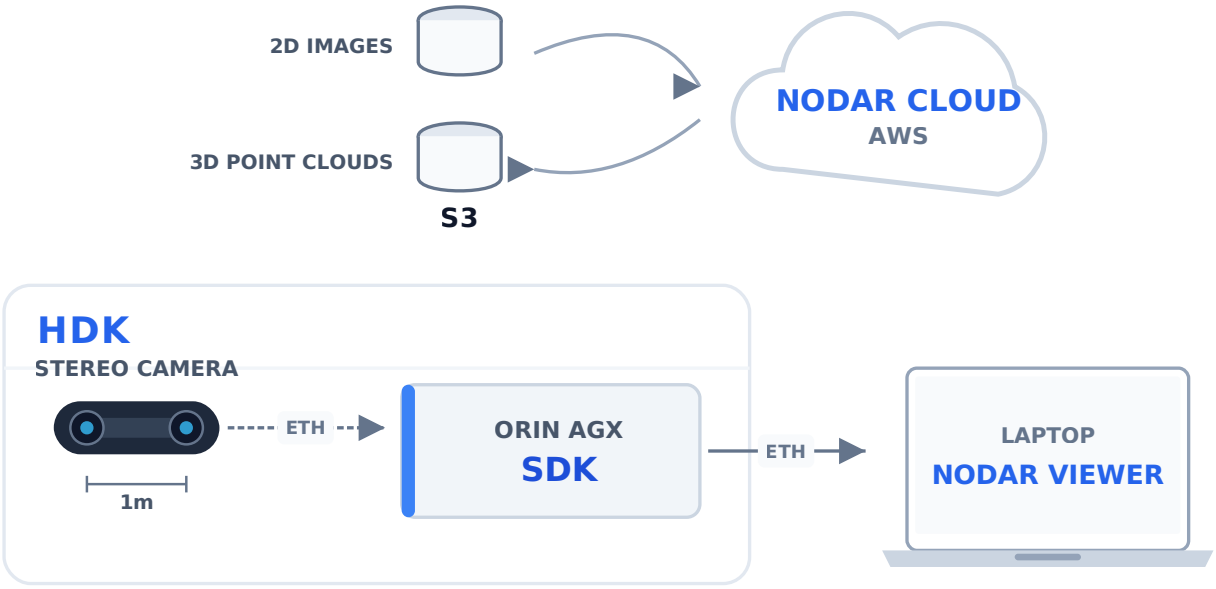
None

Table of contents

1. NODAR 3
   - 1.1 Products 3
   - 1.2 Communication Interfaces 3
   - 1.3 Additional Resources 4
   - 1.4 Contact 4
   - 1.5 Disclaimer 4
2. SDK 5
   - 2.1 SDK 5
   - 2.2 Config 11
   - 2.3 Concepts 20
3. HDK 29
   - 3.1 NODAR HDK 29
   - 3.2 Nodar HDK Quick Start Guide 30
   - 3.3 HDK Setup 32
   - 3.4 NODAR HDK Manual v2.0 37
   - 3.5 HDK Software Update 42
   - 3.6 Legal notice 43
4. NODAR Viewer 44
   - 4.1 NODAR Viewer 44
   - 4.2 Data Recorder 47
5. ROS2 48
   - 5.1 Hammerhead ROS2 48
   - 5.2 Python Examples 56
   - 5.3 C++ Examples 67
6. ZMQ 78
   - 6.1 Hammerhead ZMQ 78
   - 6.2 Python Examples 85
   - 6.3 C++ Examples 109
7. Frequently Asked Questions 138

1. NODAR

NODAR delivers high-performance stereo vision that automatically calibrates and generates depth maps. It enables wide-baseline, long-range perception for applications in automotive, robotics, and industrial systems.



1.1 Products

1.1.1 SDK

The SDK provides access to our software binaries and a library of C++ and Python examples for building your own stereo vision integration. Use it with your own compute device and cameras.

1.1.2 HDK

The Hardware Development Kit (HDK) is a complete hardware and software solution, including pre-configured cameras and compute, ready to generate high-quality point clouds out of the box.

1.1.3 NODAR Viewer

The NODAR Viewer is a powerful visualization tool for real-time stereo camera data, depth analysis, and data recording.

1.1.4 NODAR Cloud

NODAR Cloud provides cloud-based processing to match your development and AI training needs.

1.2 Communication Interfaces

ZMQ Code | ZMQ Docs

ROS2 Code | ROS2 Docs

1.3 Additional Resources

- Resources
- Application Notes and Knowledge Base

Download Offline PDF

1.4 Contact

- sales@nodarsensor.com for Questions
- support@nodarsensor.com for Support

Buy Now

1.5 Disclaimer

The information contained in the Developer Resources and connected pages is provided "as is" without any express or implied warranties of any kind, including but not limited to implied warranties of merchantability or fitness for a particular purpose. While every effort has been made to ensure the accuracy and completeness of the information, the manufacturer or publisher assumes no responsibility for errors, omissions, or inaccuracies. Specifications, features, and availability are subject to change without notice. Users are responsible for ensuring the suitability of this product for their specific application.

2. SDK

2.1 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

2.1.1 Supported Systems

Our binaries rely on Cuda and currently target Ubuntu 20.04, 22.04, and 24.04 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`
- `Cuda 12.9 / Ubuntu 24.04 / Amd 64`
- `Cuda 13.0 / Ubuntu 22.04 / Amd 64`
- `Cuda 13.0 / Ubuntu 24.04 / Amd 64`

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

2.1.2 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
```

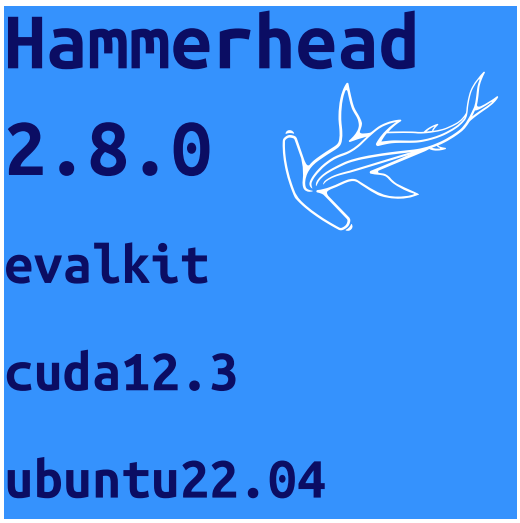
By default, our downloads will save to `${HOME}/.config/nodar/`. For example, depending on your system, the `--download` or `--install` option in `nodar-quickstart.sh` might download the following files:

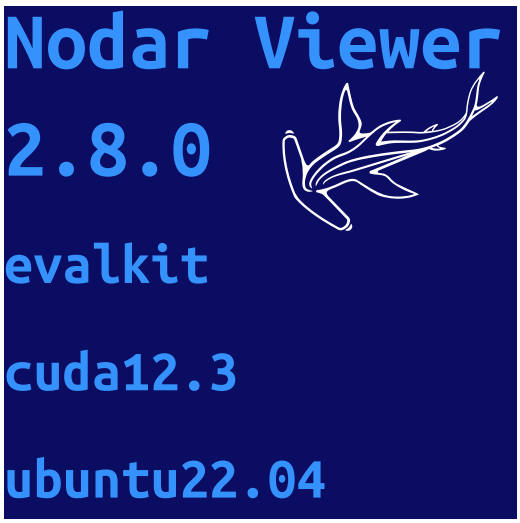
```
~$ ls ~/.config/nodar/downloads/debs/
hammerhead-2.12.1-evalkit-gd-cuda12.3-ubuntu22.04-amd64.deb
nodar_viewer-2.12.1-evalkit-gd-cuda12.3-ubuntu22.04-amd64.deb
```

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:





2.1.3 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                                                                    Priority   Status
------------------------------------------------------------
  0            /usr/lib/nodar/2.8.0-evalkit-cuda12.3-ubuntu22.04-amd64/hammerhead/nodar/hammerhead   22812302  auto mode
* 1            /usr/lib/nodar/2.9.0-evalkit-cuda12.3-ubuntu22.04-amd64/hammerhead/nodar/hammerhead   22912302  manual mode
  2            /usr/lib/nodar/2.8.0-evalkit-cuda12.3-ubuntu22.04-amd64/hammerhead/nodar/hammerhead   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                                                                       Priority   Status
------------------------------------------------------------
  0            /usr/lib/nodar/2.8.0-evalkit-cuda12.3-ubuntu22.04-amd64/nodar_viewer/nodar/nodar_viewer   22812302  auto mode
* 1            /usr/lib/nodar/2.9.0-evalkit-cuda12.3-ubuntu22.04-amd64/nodar_viewer/nodar/nodar_viewer   22912302  manual mode
  2            /usr/lib/nodar/2.8.0-evalkit-cuda12.3-ubuntu22.04-amd64/nodar_viewer/nodar/nodar_viewer   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
```

2.1.4 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

2.1.5 Getting Started

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:`

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 one of:
#   'lucid'                 - Lucid cameras (default if empty)
#   'zmq://<ipv4>:<port>'    - ZMQ topbot publisher (e.g., 'zmq://127.0.0.1:5000')
#   'ros2://<topic>'         - ROS2 topbot subscriber (e.g., 'ros2:///nodar/topbots')
camera_source = zmq://127.0.0.1:5000

# [int] Specify width and height for the camera source
# This is used when camera_source is set to zmq:// or ros2://
# This is not used if camera_source is set to lucid
camera_source_width = 2880
camera_source_height = 1860
```

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

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

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

C++ ZMQ Topbot Publisher

Python ZMQ Topbot Publisher

C++ ROS2 Topbot Publisher

Python ROS2 Topbot Publisher

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 documentation for these files here:

- Extrinsics
- Intrinsics
- Master Configuration

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.

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.

2.2 Config

2.2.1 Intrinsics Configuration File

Download Sample intrinsics.ini

The intrinsics configuration file (*intrinsics.ini*) is a simple text file used for specifying the intrinsics of the left and right cameras:

```
# Camera Intrinsics
# The following models are supported:
# 0: OpenCV Pinhole Model
# 1: OpenCV Fisheye Model
i1_model =  0
i1_fx =  5368.72291
i1_fy =  5368.72291
i1_cx =  1458.95296
i1_cy =  936.28799
i1_k1 =  -0.13332
i1_k2 =  0.98883
i1_k3 =  -5.9473
i1_k4 =  0.0
i1_k5 =  0.0
i1_k6 =  0.0
i1_p1 =  0.001
i1_p2 =  0.00053
i2_model =  0
i2_fx =  5370.75916
i2_fy =  5370.75916
i2_cx =  1431.27415
i2_cy =  935.70973
i2_k1 =  -0.13181
i2_k2 =  0.80715
i2_k3 =  -3.7122
i2_k4 =  0.0
i2_k5 =  0.0
i2_k6 =  0.0
i2_p1 =  -6e-05
i2_p2 =  -0.00035
```

CAMERA MODEL

`i1_model` **and** `i2_model`: Defines the type of camera model used.

- **0**: OpenCV Pinhole Model
- **1**: OpenCV Fisheye Model

CAMERA ASSIGNMENT

- **Camera 1 (`i1_`)**: Left camera
- **Camera 2 (`i2_`)**: Right camera

FOCAL LENGTH

`i1_fx, i1_fy` and

`i2_fx, i2_fy`

- Defines the focal length of the camera in pixels.

PRINCIPAL POINT

`i1_cx, i1_cy` and

`i2_cx, i2_cy`

- Defines the principal point (optical center) of the camera in pixels.

RADIAL DISTORTION COEFFICIENTS

`i1_k1, i1_k2, i1_k3, i1_k4, i1_k5, i1_k6` and `i2_k1, i2_k2, i2_k3, i2_k4, i2_k5, i2_k6`

Radial distortion coefficients have different meaning in the pinhole model and in the fisheye model.

Pinhole Camera Model

The radial coefficients describe how light rays deviate from an ideal pinhole projection due to lens distortion. Radial distortion causes straight lines to appear curved.

\[r_{distorted} = r \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6}, \]

where \(r\) is the radial distance from the optical center.

- `k1, k2, k3` are the primary coefficients for standard distortion correction.
- `k4, k5, k6` are higher-order coefficients for finer correction. They are useful particularly for wide FOV lenses.

Fisheye Camera Model

The fisheye model follows **Kannala-Brandt** distortion model, which differs from the pinhole model in several aspects. It incorporates only the first four distortion coefficients `k1, k2, k3, k4` . The radial distortion function is

\[\theta_{distorted} = \theta \Big(1 + k_1\theta^2 + k_2 \theta^4 + k_3 \theta^6 + k_4\theta^8\Big), \]

where \(\theta\) is the angle of the incoming ray relative to the optical axis.

The fisheye model better captures large distortions at wide angles and is best used for cameras with very wide FOV (\(FOV >100^\circ\)).

TANGENTIAL DISTORTION COEFFICIENTS

`i1_p1, i1_p2` and `i2_p1, i2_p2`

- Defines the **tangential distortion**, which occurs when the camera lens is not perfectly parallel to the image sensor.

\[x_{distorted} = x + [2p_1 xy + p_2 (r^2 + 2x^2)], \quad y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy], \]

where \((x, y)\) are the idealized undistorted coefficients and \(r^2 = x^2 + y^2\) is the radial distance from the optical center.

Note: Tangential distortion applies only to the pinhole camera model!

2.2.2 Extrinsics Configuration File

Download Sample extrinsics.ini

The extrinsics configuration file (*extrinsics.ini*) is a simple text file used for specifying the extrinsics between the cameras:

```
phi = 0
theta = 0
psi = 0
T1 = 1
T2 = 0
T3 = 0
```

The coordinate system is aligned with the optical center of the left camera and is denoted with the figure above.

The extrinsics describe the pose of the left camera relative to the right camera, described by the rotation matrix \(R\) and a translation vector \(\vec{T}\).

The rotation matrix \(R\) describes the rotation of the left camera with respect to the right camera and is assembled following \(xyz\) rotation order — \(R = R_{\psi}R_{\theta}R_{\phi}\), where the angles \(\phi\) (along x-axis) , \(\theta\) (along y-axis), \(\psi\) (along z-axis) denote Euler angles around their respective coordinate axes. In other words, the resulting \(R\) rotates the axes of the right camera to align with those of the left camera where the left-camera axes are represented in the right camera coordinates.

`phi` : Denotes the pitch or Euler angle of the left camera **in degrees** around the **x-axis** relative to the right camera

`theta` : Denotes the yaw or Euler angle of the left camera **in degrees** around the **y-axis** relative to the right camera

`psi` : Denotes the roll or Euler angle of the left camera **in degrees** around the **z-axis** relative to the right camera

The translation vector \(\vec{T} = (T_1, T_2, T_3)\) is expressed in the right camera's coordinate system and describes the translation of the right camera relative to the left camera.

`T1` : Translation in **x-direction**

`T2` : Translation in **y-direction**

`T3` : Translation in **z-direction**

You can perform the Initial Calibration routine to get the initial extrinsic parameters between the cameras.

2.2.3 Master Configuration File

Download Sample master_config.ini

The master configuration file (*master_config.ini*) is a simple text file used for configuring Hammerhead:

```
# NOTE: You must restart the app for these changes to be reloaded

# ----------------------------------------------------------------------
# Camera Source
# ----------------------------------------------------------------------

# [string] Specify camera source
# This can be either 'lucid', 'zmq://<ipv4>:<port>', or 'ros2://<topic>'.
# If empty or 'lucid', we assume that you are using Lucid cameras.
# If it starts with zmq://, then we assume you are using zmq.
# If it starts with ros2://, then we assume you are using ros2.
# An example of a valid zmq camera source is 'zmq://127.0.0.1:5000'
# An example of a valid ros2 camera source is 'ros2:///nodar/topbots'
# which we will interpret as the ros2 topic "/nodar/topbots"
# Note that the images should be topbot images, that is,
# synchronized left and right images stacked vertically
camera_source = lucid

# [string] Specify width and height for ros2 or zmq camera source.
# This is only used if camera_source is set to ros2 or zmq, not lucid
camera_source_width = 2880
camera_source_height = 1860

# [bool]
# If camera_just_stream = 1, all processing is disabled (Calibration/Stereo Matching/Grid Detect)
# and images are simply forwarded through Hammerhead as quickly as possible.
# Useful for recording and for checking that the raw image data looks correct.
camera_just_stream = 0

# ----------------------------------------------------------------------
# Lucid Settings
# ----------------------------------------------------------------------

# [string] Camera serial numbers specifying left and right camera
# Camera serial numbers can be found on the label attached to camera body
# This is only used if camera_source is set to lucid, not ros2 or zmq
camera_left_id = 231001324
camera_right_id = 231001327

# [float] Camera gain in dB. A value of -1 denotes automatic
camera_gain = -1

# [float] Camera exposure in microsecs. A value of -1 denotes automatic
camera_exposure_us = -1

# [float] Camera FPS
camera_fps = 5

# [int] Camera Mode.
# The only currently supported options are
# camera_mode = 1 (full resolution)
#   resolution      : 2880 x 1860
# camera_mode = 2 (half resolution)
#   resolution      : 1440 x  930
camera_mode = 1

# ----------------------------------------------------------------------
# Stereo Matching
# ----------------------------------------------------------------------

# [int] Stereo matching subsampling strides
# This can be 1, 2, or 4
stereo_matching_row_stride = 2
stereo_matching_col_stride = 2

# ----------------------------------------------------------------------
# Communication
# ----------------------------------------------------------------------

# [string] Communication library used for interacting with Hammerhead.
# This can be empty or one of the following case-sensitive values:
# ros2, zmq
communication_lib = zmq

# [bool] A flag indicating whether to stop hammerhead after each frame is sent and wait for a scheduler reply to continue.
wait_for_scheduler = 0

# [uint_32] If wait_for_scheduler is enabled, this field denotes the number of milliseconds to wait before continuing.
# 0 denotes "wait indefinitely".
# 1000 denotes "wait 1 second". If no scheduler reply is received in that time, then continue as if a reply was received
wait_for_scheduler_timeout_ms = 1000
```

```
# [int or comma-separated list of ints]
# Publish one or more image types using the communication lib.
# Supported types:
# 0: none
# 1: left raw image
# 2: right raw image
# 3: left rectified image
# 4: right rectified image
# 5: color blended depth image
# 6: raw topbot image (left is top-half of image, right is bottom-half)
# 7: rectified topbot image (left is top-half of image, right is bottom-half)
# 8: confidence map
# Use a comma-separated list to publish multiple types (e.g., 5,8)
# If not needed, you should specify 0 to minimize system overhead
publish_image_type = 0

# [int] If you need it, you can publish point clouds. Due to bandwidth constraints,
# you may want to subsample this point cloud. You can choose one of the following options:
# 0 : Do not publish point cloud
# 1 : Publish the full point cloud (xyz with no rgb information)
# 2 : Publish the point cloud subsampled on 3x7 tiles.
# 3 : Publish the point cloud subsampled on 5x15 tiles.
# 4 : Publish the full point cloud with rgb information
# 5 : Publish the point cloud soup (from which you can reconstruct the point cloud. This minimizes network traffic)
# If not needed, you should specify 0 to minimize system overhead
publish_point_cloud_type = 0

# ---------------------------------------------------------------------
# Navigation
# ---------------------------------------------------------------------

# [string] Navigation publisher source.
# Specify the IP address of the navigation publisher (e.g., 127.0.0.1).
# Leave empty to disable navigation data.
navigation_publisher =

# ---------------------------------------------------------------------
# Grid Detect
# ---------------------------------------------------------------------

# [bool] A flag indicating whether GridDetect will be enabled.
# Enabling this flag will enable obstacle data publishing.
enable_grid_detect = 0

# [float] limits the point cloud points positions to the specified volume
obstacle_x_min = -15
obstacle_x_max = 15
obstacle_y_min = -3
obstacle_y_max = 3
obstacle_z_min = 0
obstacle_z_max = 100

# ---------------------------------------------------------------------
# Box Filter
# ---------------------------------------------------------------------

# [bool] If enabled, the box filter with trapezoidal shape will be applied to the point cloud
enable_box_filter = 0

# [float] r_max will only be used if enable_box_filter is set to 1 and may make the other {x,y,z}_max parameters obsolete (if chosen smaller than e.g. z_max)
r_max = 1000.0

# Min z of the pointcloud, z_min is determined by the camera FOV, yaw, the baseline and the mounting height of the camera (and pitch if the camera is not mounted horizontally)
# Limit z-range further with these parameters
# [float] Min z of the pointcloud in [m]
z_min = 5.0

# [float] Max z of the pointcloud in [m]
z_max = 500.0

# x-range is determined by the camera FOV and the yaw, you can limit it further with these parameters. Increase for wide angle cameras.
# [float] Min x of the pointcloud in [m] (negative x is to the left of the camera)
x_min = -100.0

# [float] Max x of the pointcloud in [m]
x_max = 100.0

# y-limits for the trapezoidal filter. The right side top and bottom y-coordinates of the trapezoidal filter are determined by the following equations:
# y_top_right = y_top - tan(slope_top_deg * PI/180) * z
# y_bottom_right = y_bottom - tan(slope_bottom_deg * PI/180) * z
# Note that the y-axis points towards the ground
# Bottom starting point of the box filter [m], should be >=y_top
y_bottom = 10.0

# Top starting point of the box filter [m], should be <=y_bottom
y_top = -10.0

# Box filter angles measured from the z-axis
# 90 degree means straight up, 0 deg means parallel to the z-axis, -90 degree means straight down
# Slope of the box filter starting from y_bottom in degrees, should be >=-90 and <=90
slope_bottom_deg = -0.0

# Slope of the box filter starting from y_top in degrees, should be >=-90 and <=90
```

```
slope_top_deg = 0.0

# ------------------------------------------------------------------------
# Quality Assurance
# ------------------------------------------------------------------------

# [bool] If enabled, quality assurance monitoring will be performed
enable_quality_assurance = 1

# [bool] If enabled, system monitoring will be performed
enable_system_monitor = 1

# [bool] If enabled, Hammerhead monitoring will be performed
enable_hammerhead_monitor = 1

# [bool] If enabled, image monitoring (exposure, blur) will be performed
enable_image_monitor = 0

# [bool] If enabled, quality assurance findings will be published
publish_qa_findings = 0
```

For more details on specific concepts, see:

- Box Filter

2.2.4 Stereo Sensor Coordinate Configuration File

Download Sample stereo_sensor_coordinate.ini

The stereo sensor coordinate configuration file (*stereo_sensor_coordinate.ini*) defines how the stereo sensor is oriented in the world frame.

A default version of this file ships encrypted with the executable in `.deb` packages. To customize it, place your modified copy in a config directory and pass it with `-c /path/to/config/` . If `-c` is not specified, the default location that the config file should be placed is `/usr/lib/nodar/config` .

```
# Defines how the stereo sensor is oriented in the world frame.
# Using zero rotation would still work for most parts of the Hammerhead pipeline.

meters_above_ground = 0

# The world axes are defined as
#   - z-axis: parallel to the ground
#   - y-axis: pointing into the ground, to the center of Earth.
#   - x-axis: Usually to the right. It follows the right-hand rule.
# We specify the stereo sensor orientation by specifying the orientation of the left camera.
# The left camera axes are defined as OpenCV image axes with z-axis pointing into the monitor screen.
# Please provide a rotation that rotates the left camera axes to the world axes. For example, if the bar
# is tilted toward the ground, away from the horizon, then euler_x_deg should be positive.
euler_x_deg = 0
euler_y_deg = 0
euler_z_deg = 0
# The order of applying the angles. Currently supported values are "xyz" and "zyx".
order = xyz

# The IP of the source providing stereo sensor coordinates.
# If empty, then we do not listen for a source to provide the coordinates
source_ip =

# The sensor coordinates are always initialized with the aforementioned parameters.
# However, if there is a source publishing coordinate updates,
# then you also need to choose how to handle these values.
# Specifically, you need to choose one of the following blocking modes:
#
# never:  Start with the initial params. Never block main to wait for updates, but if a change is received, use it.
#         Don't use this if you are not confident in the choice of initial params above.
# first:  Same as `never`, but wait for an initial set of params to start the main thread.
#         You can/should use this if you are not confident in the choice of initial params above,
#         and if you don't expect frequent coordinate updates
# always: Before getting the next frame, make sure that we have received a parameter update.
#         If we haven't received an update, then block until we get one.
#         Use this if you expect frequent updates. Note this if updates come slowly, then you are effectively
#         throttling hammerhead main
source_blocking = first
```

MOUNTING HEIGHT

`meters_above_ground` : Height of the stereo sensor above the ground in meters.

WORLD COORDINATE SYSTEM

The world axes are defined as:

- **z-axis**: parallel to the ground, pointing forward
- **y-axis**: pointing into the ground, toward the center of the Earth
- **x-axis**: to the right, following the right-hand rule

The left camera axes follow the OpenCV image convention.

EULER ANGLES

The rotation rotates the left camera axes to the world axes.

`euler_x_deg` : Rotation around the **x-axis** in degrees. If the bar is tilted toward the ground (away from the horizon), this value should be positive.

`euler_y_deg` : Rotation around the **y-axis** in degrees.

`euler_z_deg` : Rotation around the **z-axis** in degrees.

`order` : The order of applying the Euler angles. Supported values are `xyz` and `zyx` .

EXTERNAL COORDINATE SOURCE

`source_ip` : IP address of a source providing stereo sensor coordinate updates. Leave empty to disable.

`source_blocking` : Controls how coordinate updates from the external source are handled:

- `never` : Start with the initial parameters. Never block to wait for updates, but use them if received. Only use this if you are confident in the initial parameters.
- `first` : Wait for an initial set of parameters before starting, then never block again. Use this if you are not confident in the initial parameters and don't expect frequent updates.
- `always` : Block before each frame until a coordinate update is received. Use this if you expect frequent updates. Note that slow updates will throttle Hammerhead.

2.3 Concepts

2.3.1 Box Filter

The box filter applies a trapezoidal filter to the point cloud, allowing you to define a 3D region of interest. This is useful for excluding points outside your area of interest.

Parameters

Parameter	Type	Description
`enable_box_filter`	bool	Enables/disables the trapezoidal box filter. Set to 1 to enable, 0 to disable.
`r_max`	float	Maximum radial distance from the camera in meters. Points beyond this distance are excluded. This may override other max parameters if set smaller.
`z_min`	float	Minimum z-coordinate (depth) of the point cloud in meters. Determined by camera FOV, yaw, baseline, and mounting height.
`z_max`	float	Maximum z-coordinate (depth) of the point cloud in meters.
`x_min`	float	Minimum x-coordinate in meters (negative x is to the left of the camera).
`x_max`	float	Maximum x-coordinate in meters.
`y_bottom`	float	Bottom starting point of the box filter in meters. Should be >= y_top.
`y_top`	float	Top starting point of the box filter in meters. Should be <= y_bottom.
`slope_bottom_deg`	float	Slope angle of the filter starting from y_bottom, measured from the z-axis in degrees. Range: -90 to 90.
`slope_top_deg`	float	Slope angle of the filter starting from y_top, measured from the z-axis in degrees. Range: -90 to 90.

Trapezoidal Shape

The trapezoidal shape is defined by the y-limits and slope angles. The right side coordinates are calculated as:

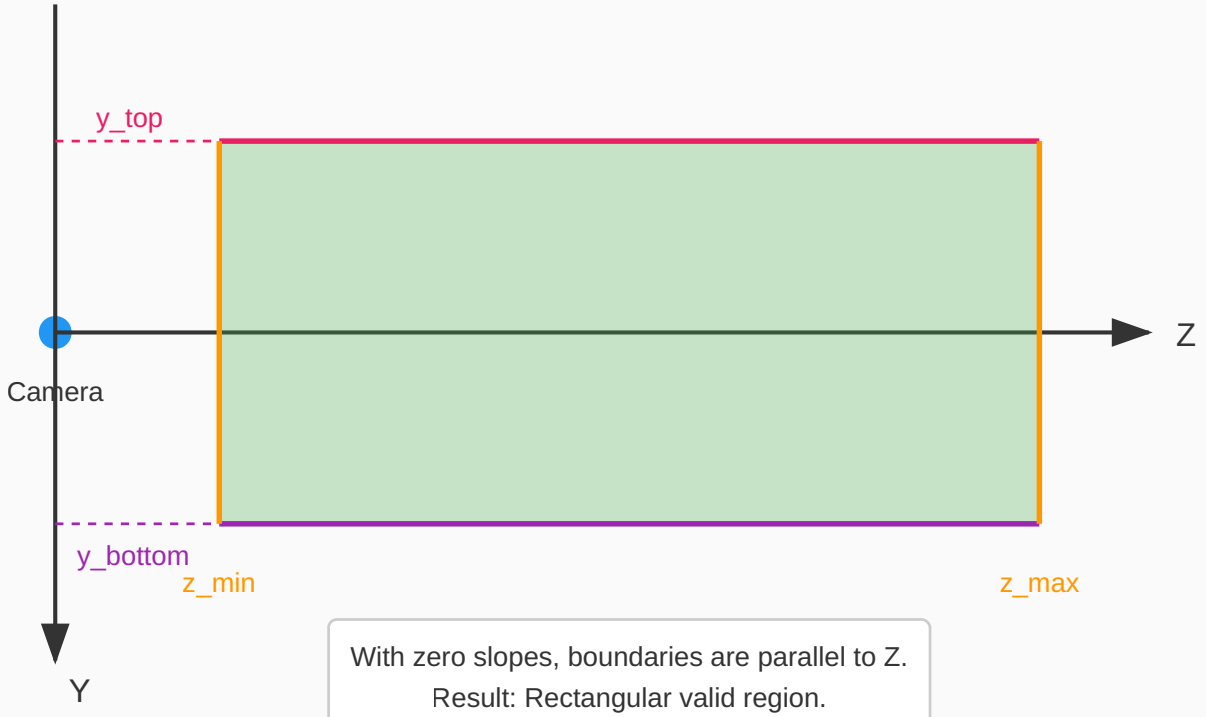
- `y_top_right = y_top - tan(slope_top_deg * PI/180) * z`
- `y_bottom_right = y_bottom - tan(slope_bottom_deg * PI/180) * z`

Note that the y-axis points towards the ground.

Examples

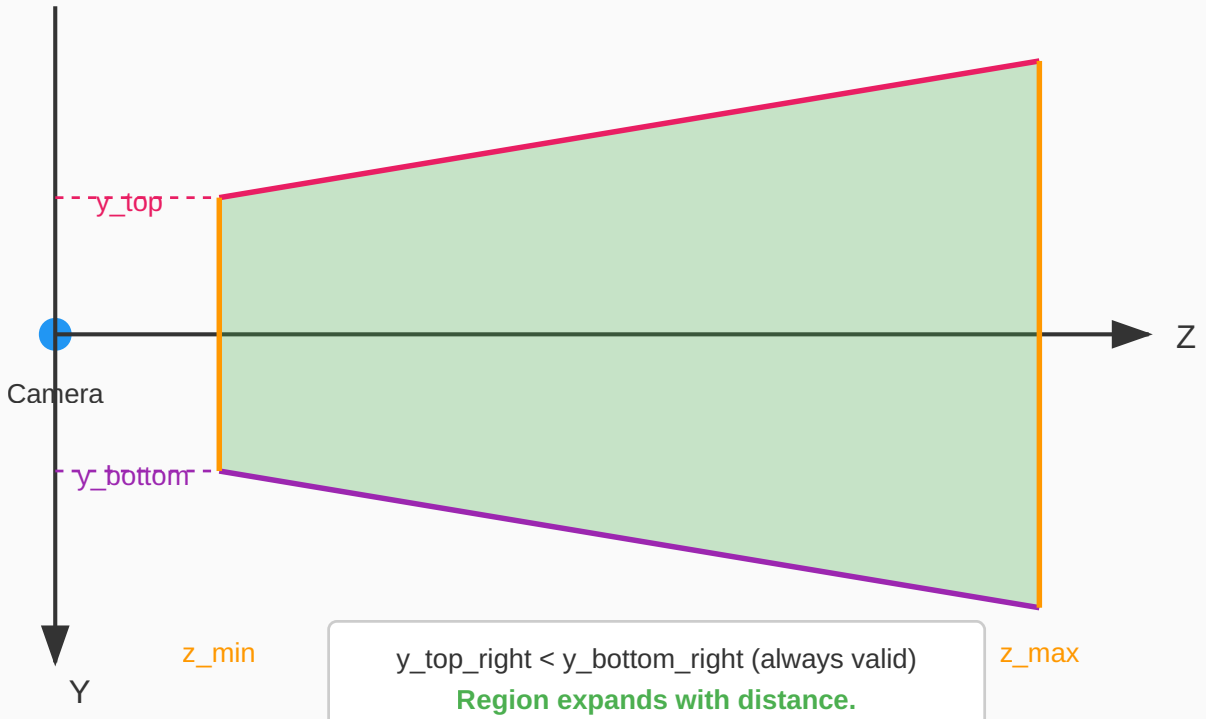
The following diagrams illustrate how different slope values affect the box filter region. When slopes cause the y_top and y_bottom boundaries to cross (crossover), points beyond the crossover are invalidated.

Default Values (slope_top_deg = 0, slope_bottom_deg = 0)



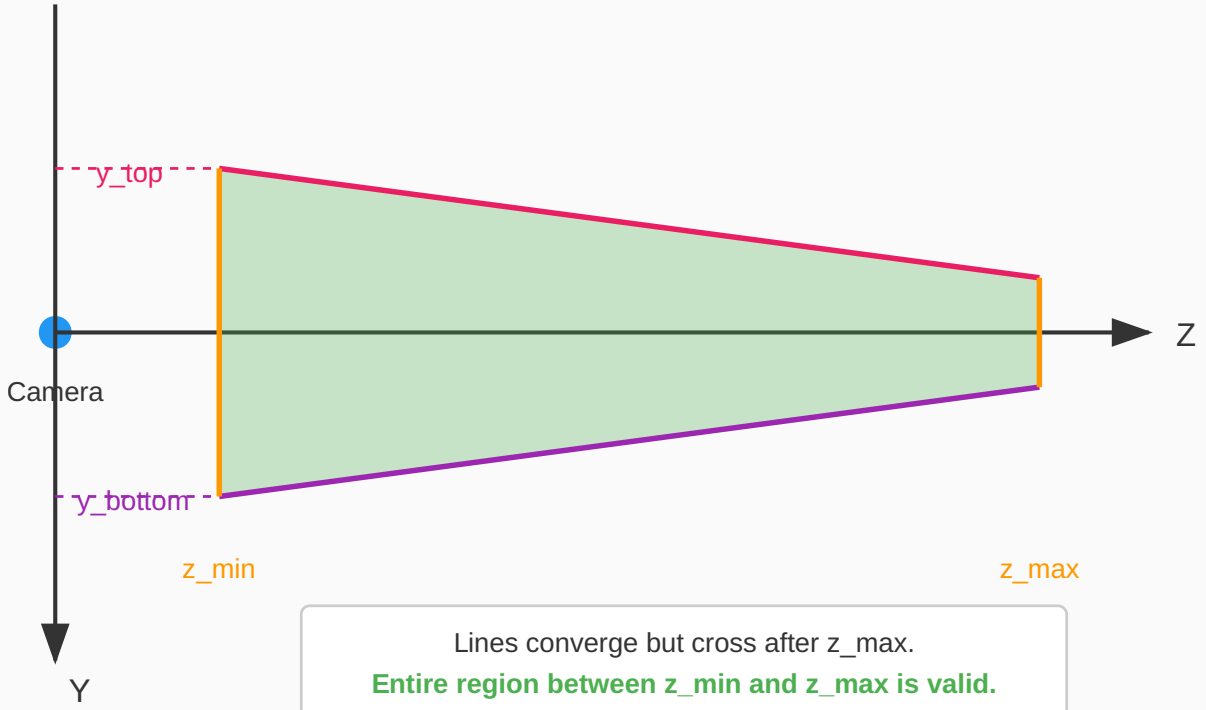
With zero slopes, boundaries are parallel to Z.
Result: Rectangular valid region.

Expanding Trapezoid (slope_top_deg > 0, slope_bottom_deg < 0)



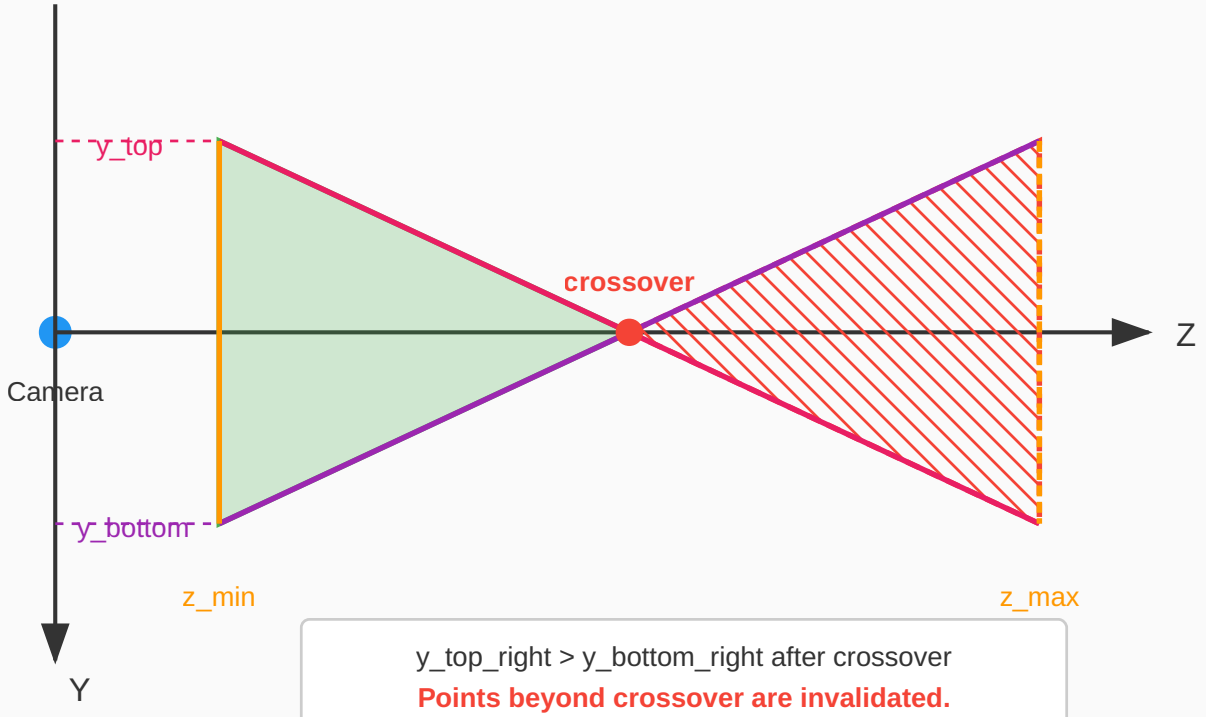
y_top_right < y_bottom_right (always valid)
Region expands with distance.

Crossover After z_max (slope_top_deg < 0, slope_bottom_deg > 0)



Lines converge but cross after z_max.
Entire region between z_min and z_max is valid.

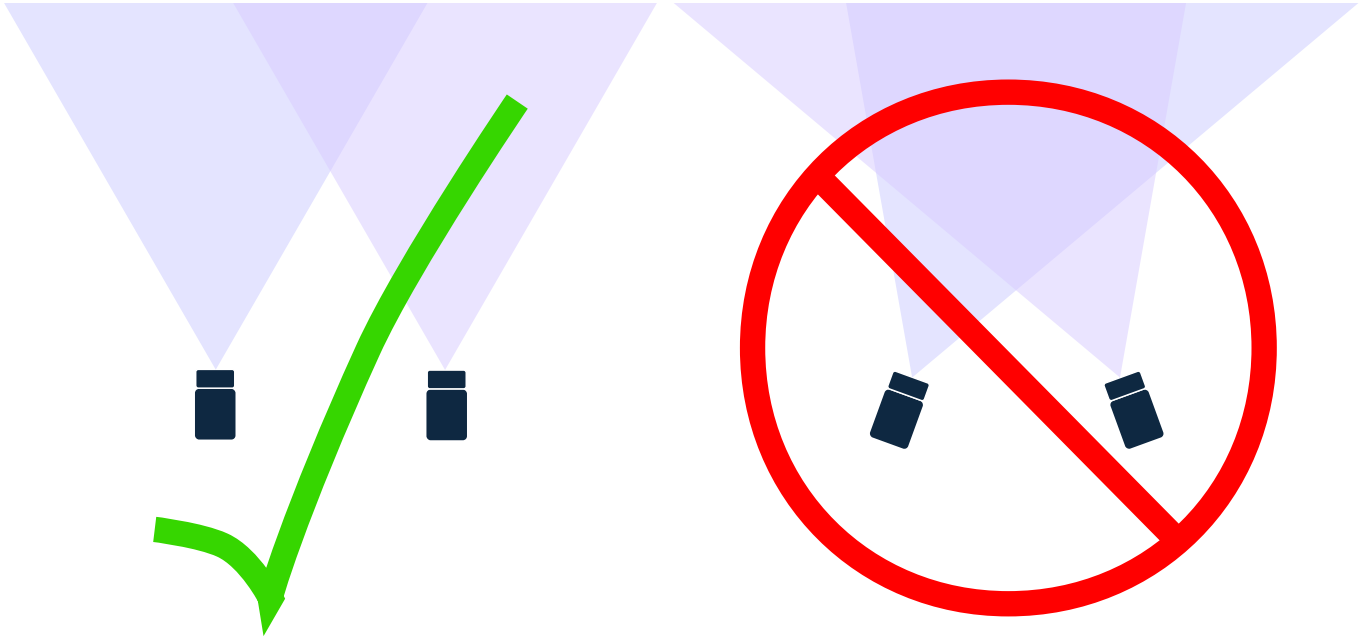
Crossover Within Range (steeper slopes)



y_top_right > y_bottom_right after crossover
Points beyond crossover are invalidated.

2.3.2 Initial Calibration

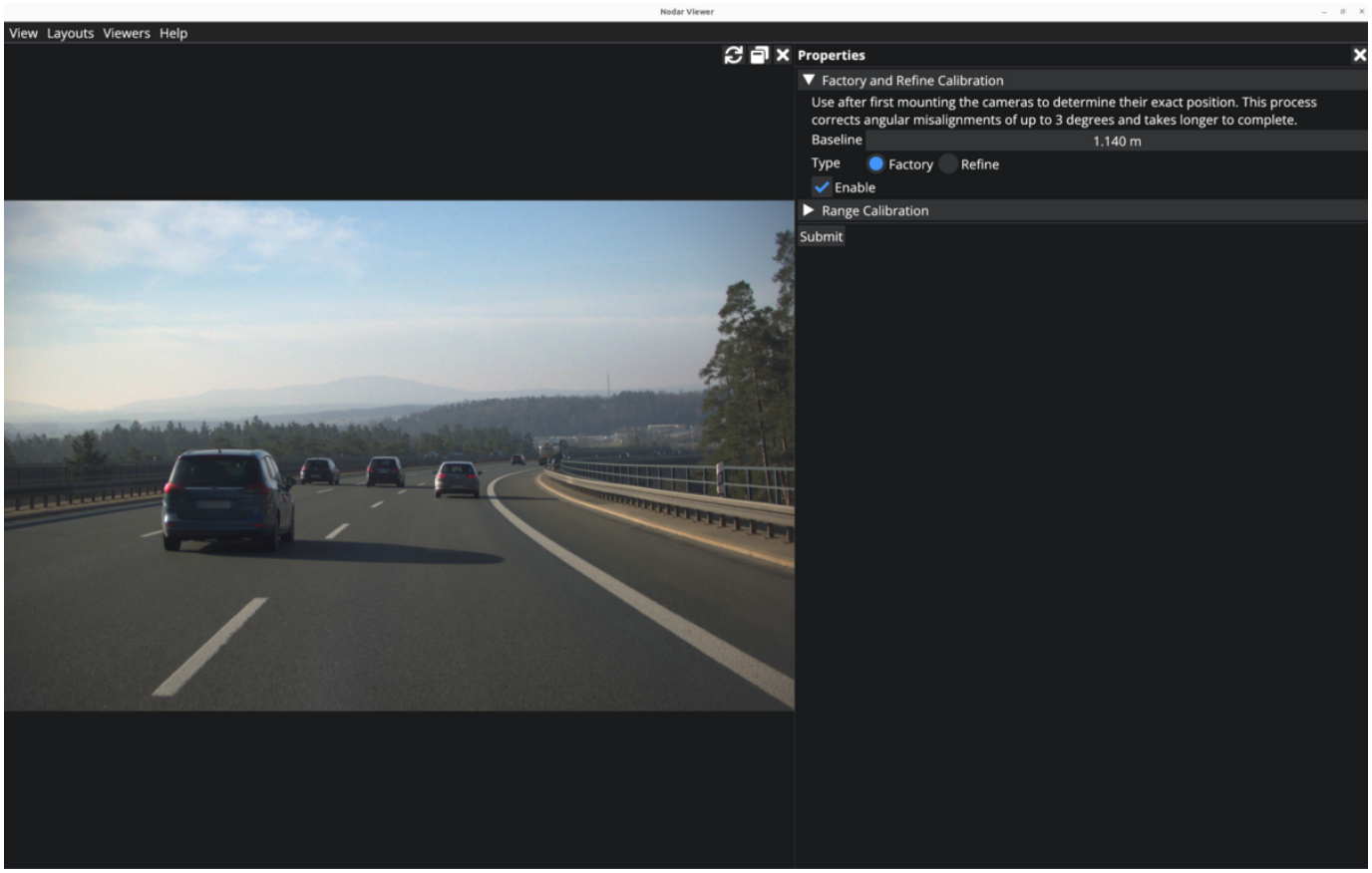
To ensure the highest probability of success during calibration, you should place the cameras in your desired configuration, and make sure that they are fastened tightly so that they do not subsequently move (which would invalidate the calibration). While placing the cameras, you must ensure that the camera axes are aligned as parallel as possible as demonstrated below. The factory calibration routine currently allows for a misalignment of up to 3 degrees.



When you are ready to calibrate your cameras in their new configuration, please move the sensor assembly outside. Static, daytime, natural scenes without reflective surfaces or moving objects work best. Avoid indoor, cluttered environments with artificial lighting, flat/texture-less surfaces, or bad weather conditions for this step. Ensure that the sensor assembly is stationary while performing calibration. Then simply run the `Nodar Viewer` app, connect it to the `Hammerhead`, open the `Initial Calibration` viewer, then open viewer's properties, choose one or several calibration options and submit your choice, as shown below. It should be ensured that the image used for calibration is devoid of observable artifacts, like reflections and motion blur. The GUI will be disabled until the calibration is done.

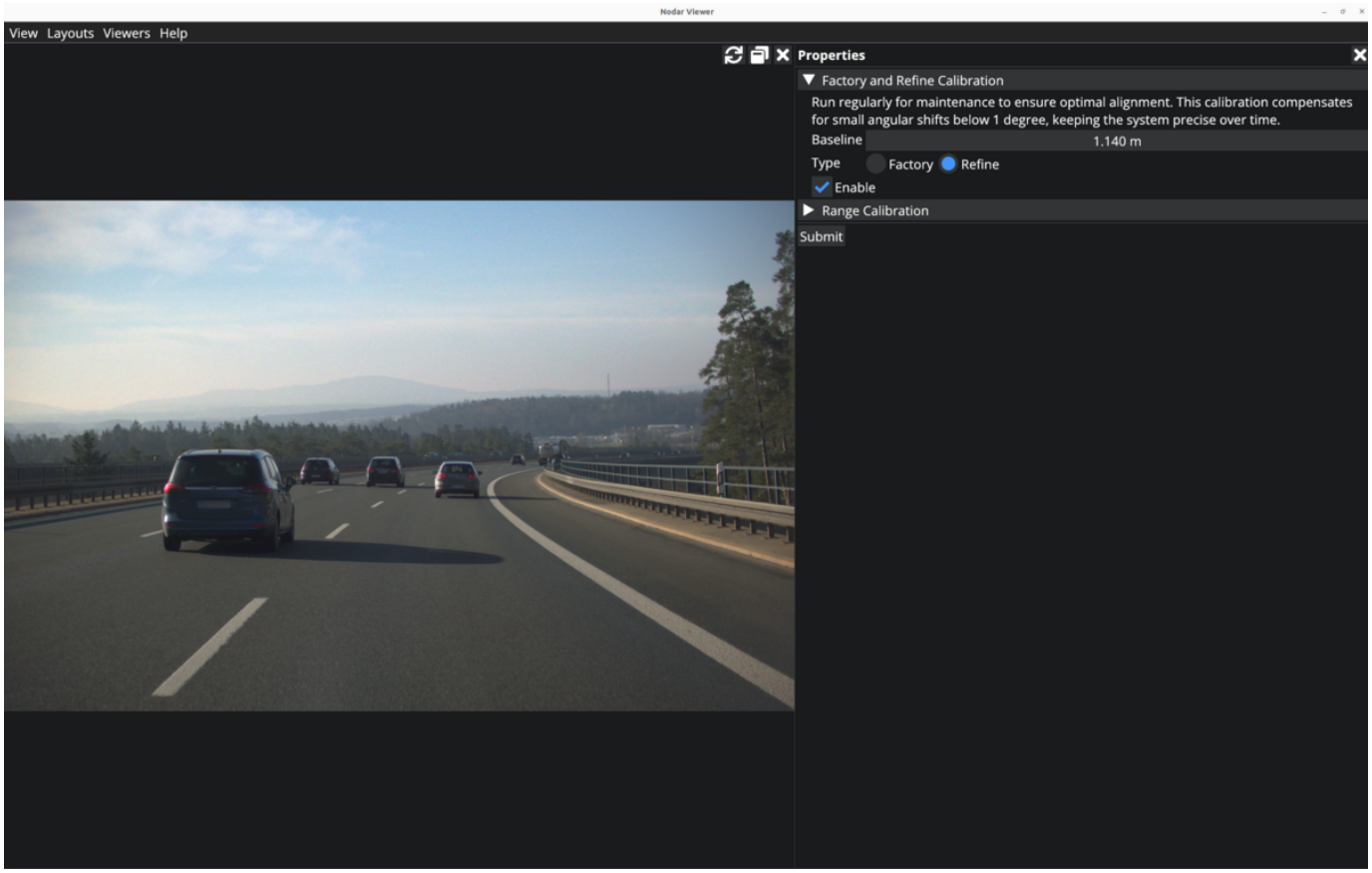
FACTORY CALIBRATION

The first time that you run the calibration manager after moving the cameras from their original factory position in the enclosure, you should run the factory calibration (a long calibration routine designed to compensate for large camera axes misalignment). To use the calibration type, first enable it by clicking the `Enable` checkbox. Enter the baseline between the cameras (in meters). After this, the `Factory` option needs to be selected. Click on `Submit` and wait for the calibration to finish.



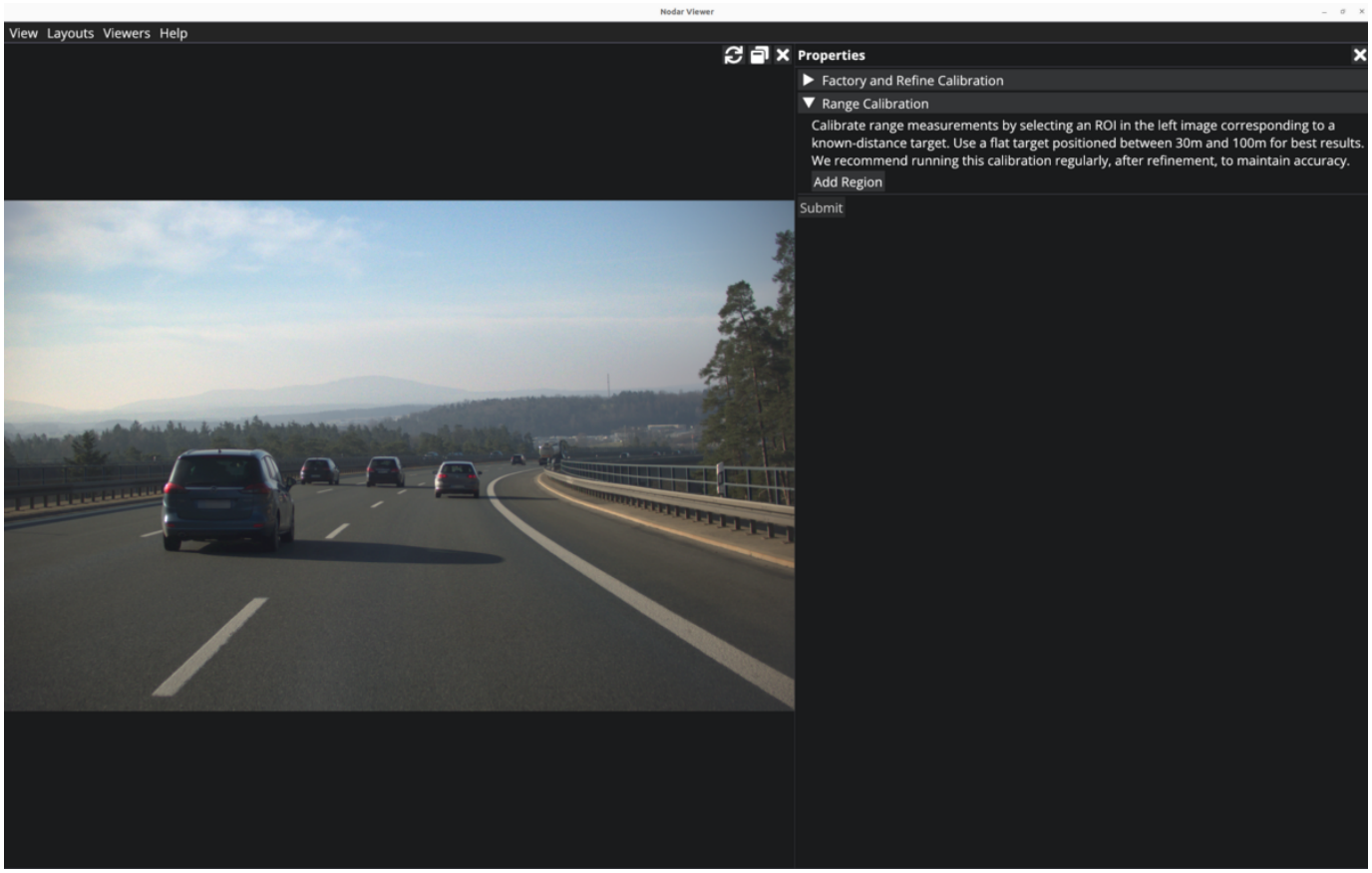
REFINE CALIBRATION

There is also an option to run a shorter, refinement calibration instead of a long factory calibration. This calibration type can be run as part of regular maintenance to maintain optimal alignment. To use the calibration type, first enable it by clicking the `Enable` checkbox. Enter the baseline between the cameras (in meters). After this, the `Refine` option needs to be selected. Click on `Submit` and wait for the calibration to finish.

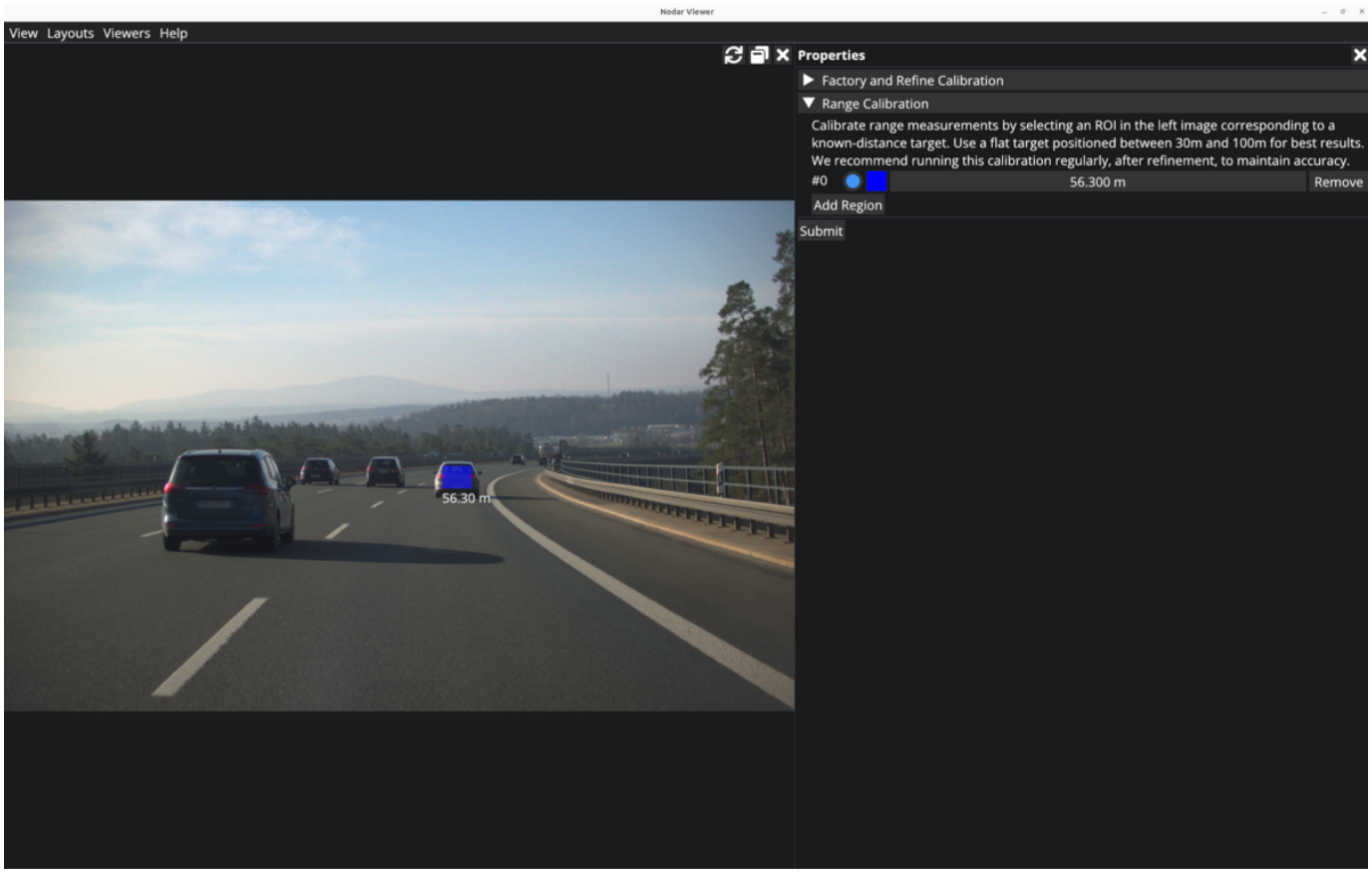


RANGE CALIBRATION

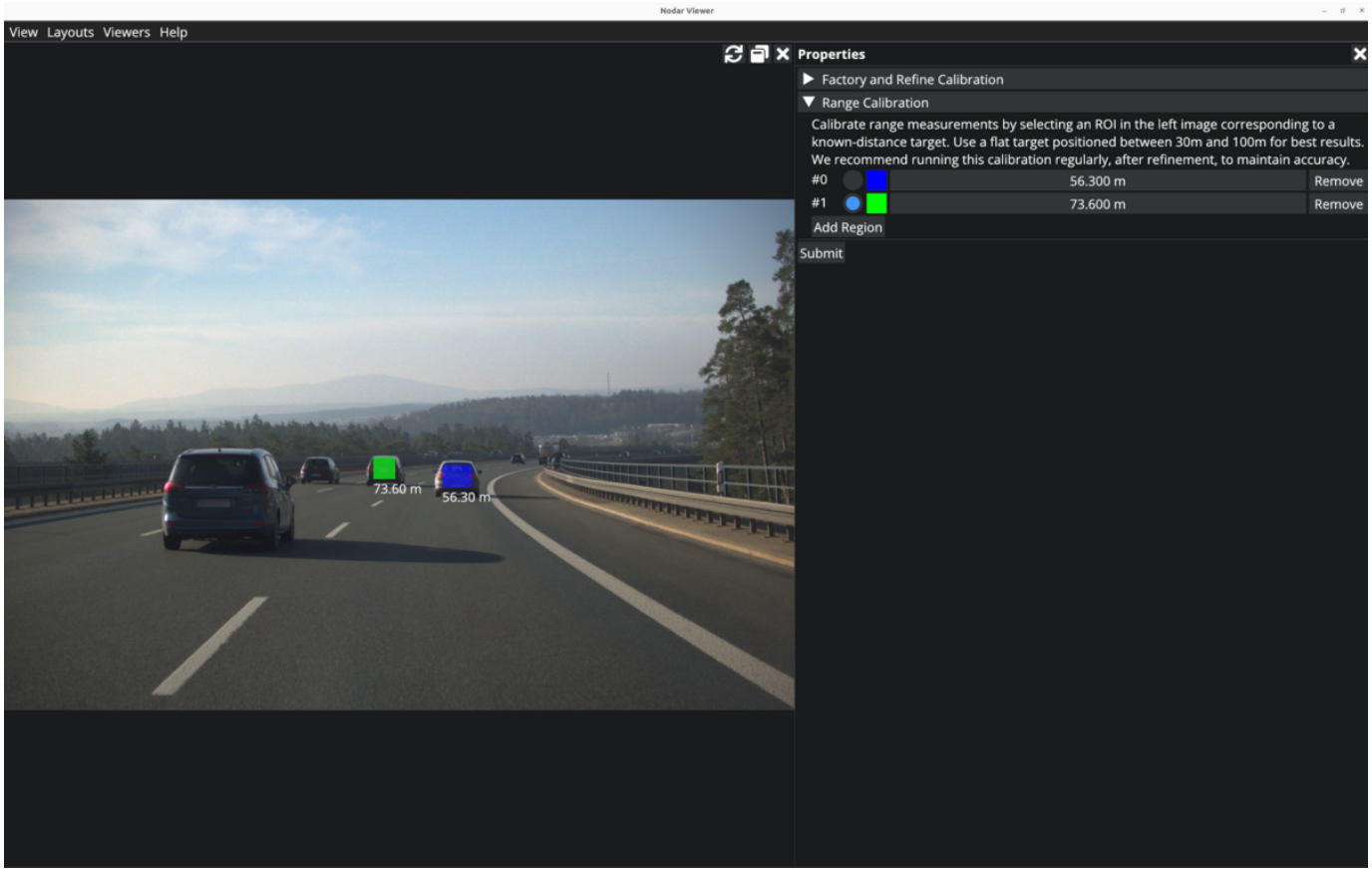
There is a third calibration option to calibrate range. This is required to be performed after any changes in the configuration of the cameras.



Click on `Add Region` which will make the viewer interactive and you can draw a rectangle. Resize the window to an appropriate size, and then select a relatively flat object near the center of the image which is sufficiently far away (>20m). This area can be selected by holding down the left mouse button and drawing on the selected object in the image. If you need to re-draw for any reason, just start to draw a new rectangle to reset your previous selection. Once you are finished selecting an appropriate area, enter the distance (in meters) to the selected object in the input field.



Optionally, you can enter multiple regions for doing the range calibration.



Press the `Submit` button after you are done selecting the regions. The distance should be measured accurately such as by using a Laser Distance Meter from the left camera. This concludes the calibration routine.

Note that factory calibration only needs to be run once after the cameras are moved to a new physical configuration. Depending on your system, this process can take a few minutes to conclude. In addition, if at any point while operating the system, you notice that the disparity estimates seem sparse or range estimates are off, even with camera axes are aligned to within 3 degrees of each other, you may try rerunning the calibration refinement and range calibration.

3. HDK

3.1 NODAR HDK

Buy Now

The NODAR Hammerhead Development Kit (HDK) is a complete hardware and software solution for generating high-quality point clouds from stereo vision cameras.

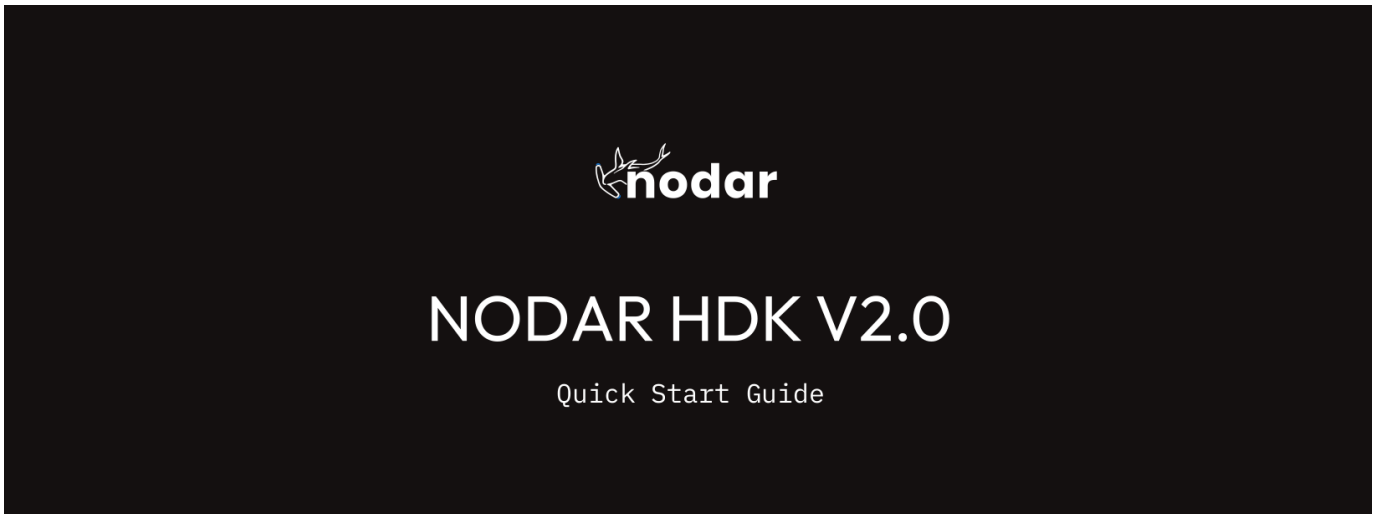
3.1.1 Getting Started

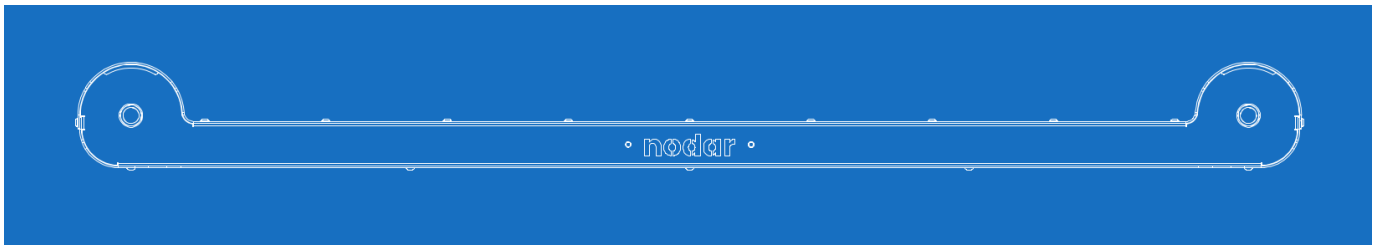
- Quick Start Guide - Get up and running in minutes
- Manual - Comprehensive documentation for setup, configuration, and operation
- Legal Notice - Patent and license information

3.1.2 Resources

- Datasheet

3.2 Nodar HDK Quick Start Guide

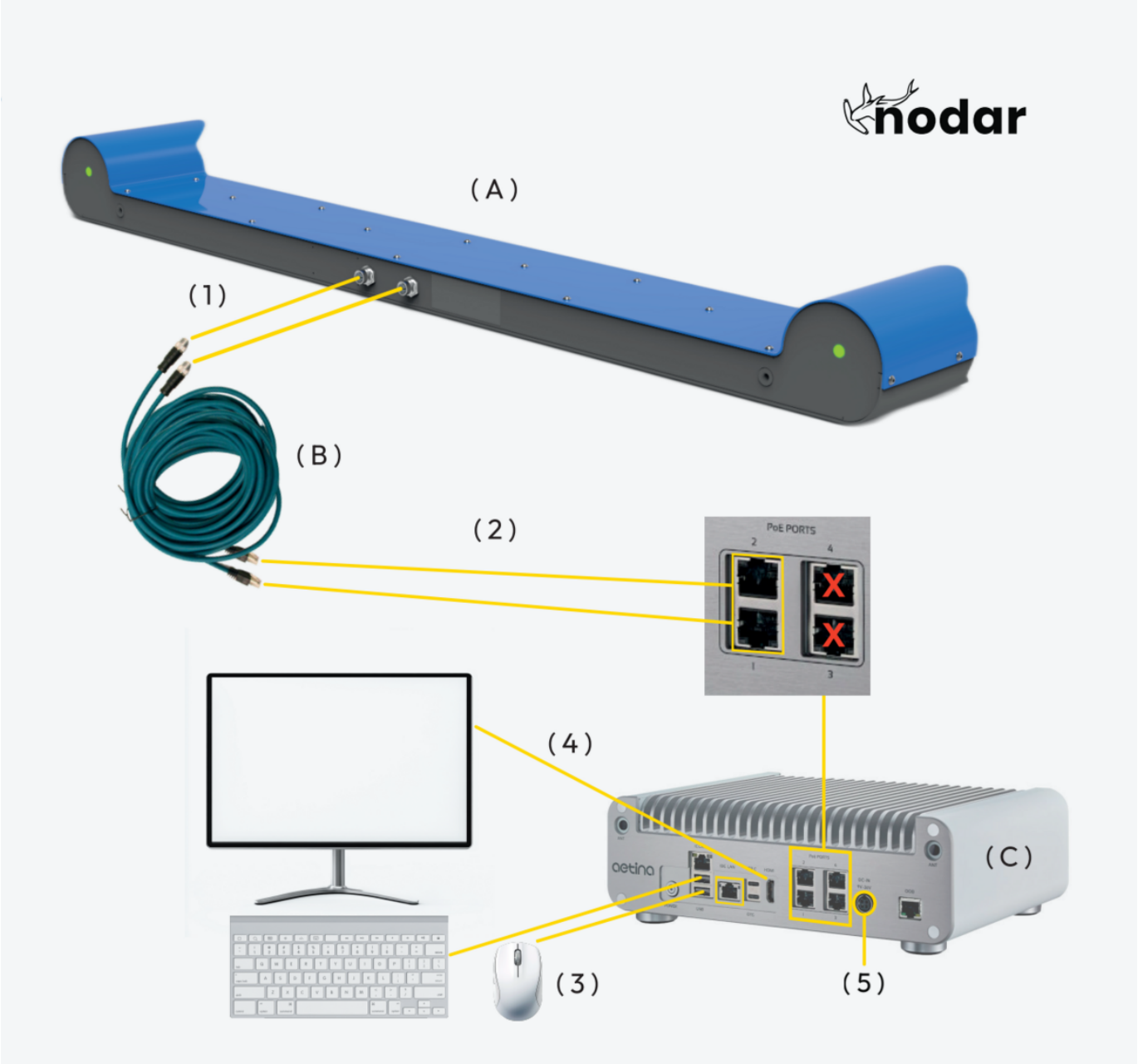




Congratulations on receiving the Nodar HDK v2.0! This kit contains:

- **(A)** HDK Camera
- **(B)** M12-RJ45 Cat6 cables
- **(C)** Orin

3.2.1 Setup Instructions



1. Connect M12-RJ45 Cables to HDK.
2. Connect M12-RJ45 Cables to PoE Ports 1 and 2.
3. Connect keyboard and mouse via USB-A.
4. Connect monitor via HDMI.
5. Connect power cable to Orin.
6. Login with password [Provided by NODAR].
7. Launch hammerhead from the terminal. Skip if autostart enabled.
8. Launch nodar_viewer from the terminal.
9. Click connect on the left side of the Viewer.
10. Wait for cameras to initialize (<1 minute).

3.3 HDK Setup

Hardware Development Kit setup scripts for configuring Linux-based hardware for real-time camera and network operations.

3.3.1 Supported Platforms

- **NVIDIA Jetson Orin AGX** - Embedded GPU computing device
- **OnLogic with Orin AGX** - Industrial edge computing platform

3.3.2 Overview

This repository provides automated setup for:

- **Background Services** - Disables unnecessary system services (updates, indexing, diagnostics) for a stable real-time environment
- **Network Configuration** - Multi-interface setup with jumbo frames (MTU 9000) for high-bandwidth camera streaming
- **PTP (Precision Time Protocol)** - Sub-microsecond clock synchronization across devices (with hardware timestamping)
- **External Time Sync** - PTP slave and PHC2SYS for synchronizing to an external PTP grandmaster (OnLogic only, opt-in)
- **Clock Optimization** - Jetson CPU/GPU clock maximization for real-time processing
- **DHCP Server** - Automatic IP assignment for connected cameras

3.3.3 Directory Structure

```
hdk_setup/
├── install.sh           # Main installation script
├── uninstall.sh         # Main uninstallation script
├── background_services/ # Disable unnecessary system services
│   └── disable_background_services.sh
├── clock/               # Jetson clock optimization
│   ├── install.sh
│   └── uninstall.sh
├── hammerhead/          # Hammerhead autostart service
│   ├── install.sh
│   └── uninstall.sh
├── mtu/                 # MTU (jumbo frames) configuration
│   ├── install.sh
│   └── uninstall.sh
├── network/             # OnLogic network & DHCP setup
│   ├── install.sh
│   ├── uninstall.sh
│   └── config/
│       ├── dhcp/
│       │   └── dhcpd.conf
│       └── netplan/
│           ├── 01-ethLAN0.yaml
│           ├── 01-ethLAN1.yaml
│           ├── 10-camera.yaml
│           └── 01-l4tbr0.yaml
├── ptp/                 # Linux PTP master setup
│   ├── install.sh
│   └── uninstall.sh
├── ptp_slave/           # Linux PTP slave setup (external time sync)
│   ├── install.sh
│   └── uninstall.sh
└── phc2sys/             # PHC to system clock sync
    ├── install.sh
    └── uninstall.sh
```

3.3.4 Installation

Jetson Devices

```
./install.sh -d jetson
```

OnLogic Devices

```
./install.sh -d onlogic
```

With Custom Camera Interfaces

```
./install.sh -d onlogic -cam_if1 ethLAN2 -cam_if2 ethLAN3
```

With External Time Sync (OnLogic only)

To synchronize time from an external PTP grandmaster via ethLAN4:

```
./install.sh -d onlogic -cam_if1 ethLAN2 -cam_if2 ethLAN3 -external-time-sync true
```

This reconfigures ethLAN4 from a camera interface to a PTP sync interface (default IP: `192.168.30.25/24`), installs the PTP slave service, and installs PHC2SYS to sync the system clock.

With External Time Sync and Custom IP

```
./install.sh -d onlogic -cam_if1 ethLAN2 -cam_if2 ethLAN3 -external-time-sync true -sync-ip 10.0.0.50/24
```

With Hammerhead Autostart

To automatically start Hammerhead on boot:

```
./install.sh -d jetson -autostart true
./install.sh -d onlogic -autostart true
```

The `-autostart` flag is `false` by default.

All Flags

Flag	Required	Default	Description
`-d`	Yes	—	Device type: `jetson` or `onlogic`
`-cam_if1`	No	`ethLAN2`	First camera interface (OnLogic)
`-cam_if2`	No	`ethLAN3`	Second camera interface (OnLogic)
`-autostart`	No	`false`	Enable Hammerhead autostart service
`-external-time-sync`	No	`false`	Enable external PTP time sync (OnLogic only)
`-sync-ip`	No	`192.168.30.25/24`	IP/CIDR for ethLAN4 when external time sync is enabled

3.3.5 Uninstallation

Jetson Devices

```
./uninstall.sh -d jetson
```

OnLogic Devices

```
./uninstall.sh -d onlogic
```

The uninstall script always attempts to clean up PTP slave and PHC2SYS services (safe no-ops if never installed) and re-enables `systemd-timesyncd`.

3.3.6 Modules

Background Services (Both platforms)

Disables unnecessary system services to ensure a stable, predictable real-time environment:

- **Update services** - apt-daily, unattended-upgrades, update-notifier, packagekit
- **Indexing services** - Tracker file indexing and metadata extraction
- **Diagnostic services** - ubuntu-report, apport crash reporting, MOTD news
- **Other** - Bluetooth, speech-dispatcher, firmware update checks

Also removes cached update notifications and suppresses future release upgrade prompts. This step is not reverted during uninstall, as these services are generally undesirable on real-time target devices.

MTU

Configures jumbo frames (MTU 9000) for high-performance data transfer.

- **Jetson**: Creates a NetworkManager dispatcher script to automatically apply settings when interfaces come up
- **OnLogic**: MTU is configured via netplan in the Network module

Network (OnLogic only)

Configures multi-interface network setup:

Default (without `-external-time-sync`):

Interface	Configuration	Purpose
ethLAN0	DHCP	Management interface
ethLAN1	Static (10.10.1.10/24)	Gateway interface
ethLAN2, 3, 4, 5	Static (10.10.x.1), MTU 9000	Camera interfaces

With `-external-time-sync true`:

Interface	Configuration	Purpose
ethLAN0	DHCP	Management interface
ethLAN1	Static (10.10.1.10/24)	Gateway interface
ethLAN2, 3, 5	Static (10.10.x.1), MTU 9000	Camera interfaces
ethLAN4	Static (192.168.30.25/24 or custom)	External PTP time sync

When external time sync is enabled, ethLAN4's MTU 9000 and DHCP subnet are removed, and the interface is reconfigured for PTP synchronization.

Also configures ISC DHCP server with subnets for camera interfaces.

PTP Master (Both platforms)

Installs and configures Linux PTP (ptp4l) for precision time synchronization:

- Operates as PTP master clock for connected cameras
- Uses E2E (End-to-End) delay mechanism
- Creates `linuxptp.service` for automatic startup and restart on failure

PTP Slave (OnLogic only, opt-in)

Configures the device as a PTP slave to synchronize time from an external PTP grandmaster. Only installed when `-external-time-sync true` is passed.

- Operates as PTP slave clock on ethLAN4 using **Layer 2 PTP** (Ethernet frames, not IP)
- The IP address on ethLAN4 is not required for PTP synchronization since Layer 2 is used. If `-sync-ip` is specified, it is recommended to be on the same subnet as the PTP grandmaster for debugging and management purposes (e.g., ping, SSH)
- Syncs to external PTP master (e.g., network grandmaster clock)
- Creates `linuxptp-slave.service` for automatic startup

WARNING: When `-external-time-sync true` is enabled, `systemd-timesyncd` (NTP) is disabled to prevent conflicts with PHC2SYS. This means the system clock is **entirely dependent on the external PTP master**. If the PTP master is unavailable, the system clock will not be synchronized and **may drift or reset to 1970**. **Always ensure a PTP grandmaster is reachable on ethLAN4 before enabling this option.**

PHC2SYS (OnLogic only, opt-in)

Synchronizes the system clock from the PTP hardware clock. Only installed when `-external-time-sync true` is passed.

- Transfers time from the PTP hardware clock (PHC) to CLOCK_REALTIME
- Runs after PTP slave has synchronized with the external master
- Creates `phc2sys.service` for automatic startup
- `systemd-timesyncd` is disabled during install to give PHC2SYS sole control of the system clock (re-enabled on uninstall)

Clock (Both platforms)

Maximizes CPU/GPU clocks for optimal real-time performance:

- Sets maximum power profile (`nvpmodel -m 0`)
- Runs `jetson_clocks` for maximum CPU, GPU, and EMC (memory) frequencies
- Maximizes VIC (Video Image Compositor) frequency if available
- Automatically restores default clocks on shutdown

Hammerhead Autostart (Optional)

Creates a systemd service to automatically start Hammerhead on boot:

- Runs as the user who installs the service (to access user config files)
- Starts after network, DHCP, and PTP services are ready
- When external time sync is enabled, also waits for PTP slave and PHC2SYS services
- Automatically restarts on failure
- Updates journald log level for debug output visibility

Managing the Hammerhead service (when installed with `-autostart true`):

```
# Check service status
sudo systemctl status hammerhead

# Start / stop / restart
sudo systemctl start hammerhead
sudo systemctl stop hammerhead
sudo systemctl restart hammerhead

# Follow logs in real time
sudo journalctl -u hammerhead -f
```

3.3.7 Requirements

- Linux (Ubuntu/Debian-based)
- sudo privileges (scripts invoke sudo internally as needed)
- For Jetson: NVIDIA Jetson Orin AGX with JetPack
- For OnLogic: OnLogic with Orin AGX and multiple Ethernet interfaces

3.3.8 Services Installed

- `linuxptp.service` - PTP master clock synchronization (both platforms)
- `linuxptp-slave.service` - PTP slave clock synchronization (OnLogic, when `-external-time-sync true`)
- `phc2sys.service` - PHC to system clock sync (OnLogic, when `-external-time-sync true`)
- `clocks.service` - Clock maximization at startup (both platforms)
- `clocks-restore.service` - Clock restoration on shutdown (both platforms)
- `isc-dhcp-server` - DHCP server for camera networks (OnLogic)
- `hammerhead.service` - Hammerhead autostart (optional, both platforms)

3.4 NODAR HDK Manual v2.0

3.4.1 1. Introduction

The NODAR Hammerhead Development Kit (HDK) generates high-quality point clouds from a pair of cameras. It automatically calibrates and generates depth maps for a stereo vision camera setup. The HDK allows you to utilize high-resolution cameras, simplifies calibration, and enables wide-baseline, long-range stereo vision. This document assumes that the two provided lens-camera pairs are parallel and facing forward (along the direction of vehicle motion).

1.1 Purpose and Scope

This document is intended to guide the initial setup and first run of your HDK. It includes information on the components of the system and instructions on how to perform system assembly, configure system parameters, launch the HDK application, and record data for diagnostic evaluation.

The **Hammerhead Development Kit** is referred to as **HDK** in this document to improve readability.

1.2 Audience

This document is intended for trained engineers tasked with integrating, testing, or evaluating the system. Users are assumed to possess an operational understanding of camera-based vision systems, familiarity with network configuration, and terminal usage in Unix environments.

It is highly recommended that the user read and understand this document in its entirety prior to installation and operation of the system.

3.4.2 2. Unboxing

2.1 What's in the Box?

The NODAR HDK comes partially assembled and is ready for testing within minutes. The figure below demonstrates the components of a typical kit. Your kit may vary slightly from this configuration.

Carefully unpack the case and ensure that all the components mentioned in the packing list are present in the kit. A typical kit will contain the items listed in Table 1.

Item	Quantity
Hammerhead Sensor	1
M12-to-RJ45 IP67 Cat6a Cable, 15.0m	2
Aetina Onlogic Industrial Orin AGX	1
Orin Power Cable	1
Ethernet cable, CAT 8, 10 ft	1

Table 1: System Components

2.2 Handling and Operation

The HDK hardware contains sensitive electronics, optical elements, and glass components that need to be handled carefully. Please connect all cables before applying power to the system. Disconnecting any cables while the system is powered on may damage the system. Before operation, visually inspect the system for loose cables, debris on the optical elements, or any other damage to the kit's components. Tie down any loose cables in a tension-released manner to prevent tripping hazards and avoid exerting excess force on any connectors while the vehicle is in motion.

The sensor assembly is designed to be operated outdoors on natural scenes while mounted on a vehicle. However, initial setup and verification may be performed indoors if necessary. Note that the system works best on natural scenes, and the quality of the results may be sub-optimal in indoor office environments with poor lighting or shiny/texture-less surfaces. For best results while setting up indoors, point the cameras through a window at any natural scene.

3.4.3 3. Assembly

This section outlines the procedure for setting up the HDK hardware. The entire assembly should be initially tested as shipped to verify that everything is in working order. As outlined below, the sensor assembly may also be mounted using additional hardware.

3.1 Connecting Cables

For cable connection instructions, please refer to the Quick Start Guide.

3.2 Cleaning and Modifications

The sensor assembly, comprised of the framing rail, sensors, and mounts, consists of IP67-rated components and can be mounted on the exterior of a vehicle. During regular operation dust, debris, or moisture may accumulate on the IP67 lens tubes resulting in a degradation of output data. It is recommended that the lens tubes be wiped with a microfiber cleaning cloth before operating the system whenever this happens.

3.3 Mounting

The Hammerhead sensor has multiple 1/4-20 mounting points on the back and bottom. These can be used with most standard camera mounts. When mounting on a vehicle or any situation where the Hammerhead sensor may move, we recommend attaching eye-bolts and running a strap or other connector through the eye-bolts as an additional safety measure.

3.4.4 4. Application Configuration, Launch, & Interaction

The compute for this system is the Aetina Onlogic Industrial Orin AGX. The HDK software comes pre-installed on the Orin included with the kit. For initial setup, it is necessary to connect a mouse, keyboard, and display to the Orin. Following initial setup, the HDK can be configured to run in headless mode for data collection if required.

You can launch Hammerhead from the terminal with `hammerhead`

You can launch Nodar Viewer from the terminal with `nodar_viewer`

4.1 Nodar Viewer

The Nodar Viewer is optimized for visualizing the output from Hammerhead, providing a way to analyze and interact with Hammerhead. Designed with a focus on low latency and high-speed operation, the Nodar Viewer ensures smooth performance. The Viewer also offers the ability to play back recorded data.

The Viewer can be run directly on the Orin for quick viewing and system verification. However, for data collection and long-term operation, it is recommended to install the Viewer on a separate computer and connect to the HDK over the network (see Connecting to a Computer). This approach offloads visualization and recording tasks from the Orin, ensuring optimal performance.

4.2 Communication interface

The Hammerhead Development Kit (HDK) offers several communication options enabling integration into diverse applications.

4.3 Calibration

The HDK is factory-calibrated before delivery.

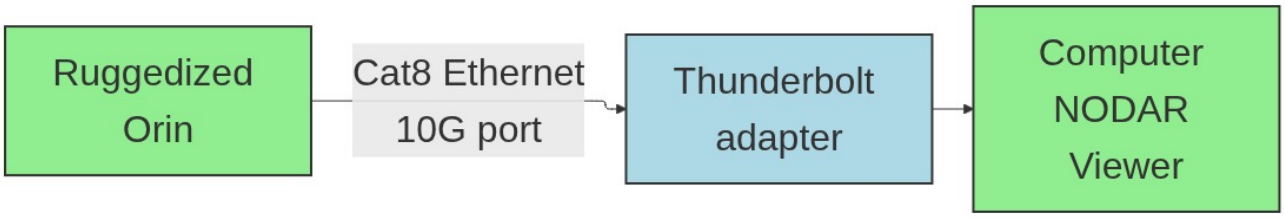
For a detailed description of the initial calibration routine, please consult Initial Calibration.

3.4.5 5. Connecting to a Computer

For data collection and long-term operation, it is recommended to connect the HDK to a separate computer and run the Viewer there. This offloads visualization and recording tasks from the Orin, ensuring optimal performance.

To connect the HDK to a computer, use the 10G port of the ruggedized Orin.

The 10G port of the ruggedized Orin has a static IP of `10.10.1.10` (if you have a HDK version greater than 2.4.2)



In case you have a 10G port on your computer, you can directly connect it to the ruggedized Orin. In case you do not have a 10G port but have a Thunderbolt port, you can use a Thunderbolt adapter such as OWC

You should configure your computer's Ethernet interface with a static IP address in the `10.10.1.X` range (e.g., `10.10.1.100`), ensuring it is on the same subnet as the ruggedized Orin (`10.10.1.10`).

Make sure you use high quality Cat8 Ethernet cables which are meant for high bandwidth data transmission.

The easiest way to interact with the HDK is through the Viewer. You can download the Viewer on your computer by following instructions here.

3.4.6 6. Data Collection on the Computer

When collecting data with the HDK, it is recommended to start and stop the recording directly from the Viewer on your computer. This ensures proper synchronization and minimizes the risk of dropped frames.

To achieve the most reliable data capture:

- Close all open Viewer tabs except the Data Collection tab before starting the recording. Visualizing data (e.g., live point clouds) during capture can consume significant system resources and result in frame drops.
- Record only the data you actually need for your session. This helps conserve storage space, and optimizes bandwidth.
- Make sure your network connection to the ruggedized Orin is stable before starting collection.
- Use a high-performance SSD for recording to avoid write-speed bottlenecks that can cause frame drops.
- Verify that your storage device on the computer has enough free space for the planned capture session.

By keeping the Viewer focused solely on the Data Collection tab and avoiding unnecessary live visualization, you maximize the integrity and performance of your recordings.

3.4.7 7. Legal Notice

The Hammerhead Development Kit (HDK) is protected by a range of patents. For details on patent protection and software licenses, please refer to Legal Notice.

3.4.8 8. Frequently Asked Questions (FAQ)

The Frequently Asked Questions and Troubleshooting Guide is a growing collection of answers to the most common questions users encounter when working with the HDK. This resource is regularly updated and can be found on the FAQ page.

3.4.9 9. Contact Us

If you need support from our support engineers, please send us an email at support@nodarsensor.com.

3.5 HDK Software Update

3.5.1 1. Download the quickstart script

```
curl -O https://docs.nodarsensor.net/sdk/nodar-quickstart.sh
chmod +x nodar-quickstart.sh
```

3.5.2 2. Run the quickstart script

```
./nodar-quickstart.sh <uuid> --install
```

Replace `<uuid>` with your customer-specific identifier.

3.5.3 3. Manual install (if auto-install fails)

If the automatic installation does not succeed, you can manually download and install the packages.

Download the Hammerhead and Nodar Viewer `.deb` files from:

```
https://downloads.nodarsensor.net/<uuid>
```

Then install them:

```
sudo apt install ./<path_to_hammerhead_deb>
sudo apt install ./<path_to_nodar_viewer_deb>
```

Note: If your HDK is running version 2.11.4 or earlier, please follow the legacy update instructions.

3.6 Legal notice

3.6.1 1. Third Party License Agreement

Information on third-party software licenses can be found under Third Party Software License Agreements - Hammerhead for Hammerhead software and Third Party Software License Agreements - Viewer for the viewer.

3.6.2 2. Patent Protection

The following product is protected by patents in the U.S. and elsewhere. This page is provided to satisfy the virtual patent marking provision of various jurisdictions including the virtual patent marking provisions of the American Invents Act and provide notice under 35 U.S.C. § 287(a). The following list of products and patents may not be all inclusive. For example, some products listed here may be covered by patents in the United States and elsewhere that re not listed, and other products not listed here may be protected by one or more patents in the United States and elsewhere. The following list of products may be covered by one or more of the following U.S. Patents.

Product Name	Patent Number(s)	Pending Application(s)
Hammerhead	U.S. 11,282,234	U.S. 18/500,714
Hammerhead	U.S. 11,321,875	Europe 21744461.1
Hammerhead	U.S. 11,321,876	Korea 10-2023-7021904
Hammerhead	U.S. 11,427,193	India 202217044854
Hammerhead	U.S. 11,834,038	India 202217056308
Hammerhead	U.S. 11,983,899	
Hammerhead	U.S. 12,094,144	
Hammerhead	U.S. 12,125,215	
Hammerhead	U.S. 12263836	
Hammerhead	Japan 7319003	
Hammerhead	Japan 2022-544695	
Hammerhead	Japan 2022-561423	
Hammerhead	Japan 2023-1114397	
Hammerhead	Korea 10-2550678	
Hammerhead	Korea 10-2583256	
Hammerhead	Korea 10-2022-7028954	
Hammerhead	Korea 10-2023-7021904	
Hammerhead	Korea 10-2023-7032546	

4. NODAR Viewer

4.1 NODAR Viewer

NODAR Viewer is a powerful and intuitive software tool designed to visualize and control stereo camera data. Tailored for professionals working with stereo vision systems, NODAR Viewer lets users view real-time camera streams, analyze depth data, and adjust camera settings. With its user-friendly interface and robust features, NODAR Viewer empowers users to unlock the full potential of stereo cameras in various applications, from robotics to automotive systems.

The viewer can be run on the Orin running Hammerhead or a separate computer connected to the Orin via 10G Ethernet. If run on the Orin that also runs Hammerhead, both the viewer and Hammerhead will compete for GPU resources and the frame rate will drop.

Data can be viewed in real-time from Hammerhead or replayed from disk.

4.1.1 Downloads

Download for Ubuntu x86

Download for Ubuntu ARM

4.1.2 Sample Datasets

Download Short Test Data

Download Full Sample Data

4.1.3 How to Use NODAR Viewer

User Interface Documentation

4.1.4 Minimum System Requirements

Processor (CPU)

- 3.0 GHz or higher clock speed
- Minimum 4 physical cores (8 threads)

Graphics Card (GPU)

- Minimum 2 GB VRAM
- Vulkan 1.3 support required

Memory (RAM)

- Minimum: 16 GB DDR4
- Recommended: 32 GB for large datasets or concurrent workloads

Storage

- Minimum 256 GB SSD
- NVMe SSD recommended for improved read/write speeds

4.1.5 Installation

To install the viewer:

```
sudo apt install ./nodar_viewer-<arch>.deb
```

The software will be installed to `/usr/bin/nodar_viewer`.

To uninstall:

```
sudo apt remove nodar_viewer-<version>-evalkit-gd-<arch>
```

4.1.6 Playing Back Sample Data

1. Open NODAR Viewer by typing `nodar_viewer` in the terminal.
2. Go to **View → Properties → Viewer Mode** and select **Playback** from the dropdown menu.
3. Click **Open** and point to the sample dataset you downloaded.
4. Click **Validate**.
5. Click the **Play** icon to start playback.

4.1.7 Version Archive

Previous Versions

Version	Ubuntu ARM	Ubuntu x86
2.8.0	Download	Download
2.7.1	Download	Download
2.6.2	—	Download
2.4.4	Download	Download
2.4.2	Download	Download
2.3.0	Download	Download
2.0.9	Download	Download
2.0.7	—	Download

4.2 Data Recorder

The following data can currently be recorded through the Viewer.

A `-` in the Viewer Field column indicates that the data is always saved with every recording. Topbot, Disparity, and Left Rectified are enabled by default in the Viewer.

Viewer Field	File/Folder	Description
-	`config/master_config.ini`	The master configuration file used for configuring Hammerhead. See Master Configuration for details.
-	`config/intrinsics.ini`	Intrinsics file for the left and right cameras. See Intrinsics for details.
-	`config/extrinsics.ini`	Extrinsics between the left and right camera. See Extrinsics for details.
-	`config/version.ini`	Version of the software.
-	`details`	Stores metadata in YAML format. These YAML files provide additional information about the dataset, such as timestamps, reprojection matrix and the disparity/rectified to raw camera rotation matrix.
Topbot	`topbot`	Contains the raw colored left and right images stacked vertically; saved in TIFF format.
Disparity	`disparity`	Contains disparity maps in unsigned 16-bit format. They are saved as uncompressed TIFF files.
Left Rectified	`left-rect`	Left rectified images; saved in uncompressed TIFF format.
Confidence	`confidence-map`	Stores the confidence map in TIFF format.
Depth	`depth`	Stores the depth image in TIFF format.
Color Depth Map	`depth-colormap`	Stores the color blended depth map in TIFF format.
Grid Detect Data	`tracked-objects`	Contains YAML files with ID, bounding box, velocity and occupancy cells for objects in the scene. This is available only with Grid Detect enabled.
BEV Image	`obstacle-bev-colored`	Stores the BEV view in TIFF format.
Grid Detect Image	`obstacle-detector`	Stores the visualization of the Grid Detect Data in TIFF format. This is available only with Grid Detect enabled.
Occupancy Map	`occupancy-map` & `occupancy-data`	Stores the binary occupancy map and metadata in TIFF format. This is available only with Grid Detect enabled.

5. ROS2

5.1 Hammerhead ROS2

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

5.1.1 Table of Contents

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

5.1.2 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

You can also use the Topbot Publisher examples to learn how to send messages to Hammerhead for processing.

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

5.1.3 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

# IMPORTANT!: Load the DDS transport config
# See "DDS Transport Configuration" section below.
# If you are seeing bad throughput over UDP, check your buffer sizes:
#     sysctl net.core.rmem_max net.core.wmem_max
export FASTRTPS_DEFAULT_PROFILES_FILE=$(pwd)/config/fastdds.xml

# 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
```

> Correct DDS configuration is one of the most important steps you can take to get Ros2 running smoothly. Using the Ros2 defaults, a system that is capable of 20fps can see performance drop under 1fps! See "DDS Transport Configuration" section below on how to tune this.

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
```

5.1.4 Message Types & Topics

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

Image Topics

Topic	Description	Message Type
`/nodar/left/image_raw`	Raw left camera feed	`sensor_msgs/Image`
`/nodar/right/image_raw`	Raw right camera feed	`sensor_msgs/Image`
`/nodar/left/image_rect`	Rectified left image	`sensor_msgs/Image`
`/nodar/right/image_rect`	Rectified right image	`sensor_msgs/Image`
`/nodar/disparity`	Disparity map (Q12.4 format)	`sensor_msgs/Image`
`/nodar/color_blended_depth/image_raw`	Color-coded depth visualization	`sensor_msgs/Image`
`/nodar/topbot_raw`	Raw topbot image (left is top-half, right is bottom-half)	`sensor_msgs/Image`
`/nodar/topbot_rect`	Rectified topbot image (left is top-half, right is bottom-half)	`sensor_msgs/Image`
`/nodar/confidence_map`	Confidence map	`sensor_msgs/Image`

3D Data Topics

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

Control Topics

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

5.1.5 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.

5.1.6 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

CAMERA SOURCE EXAMPLES

- **Topbot Publisher** - Publish stereo topbot images from disk to Hammerhead over ROS2

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

CAMERA SOURCE EXAMPLES

- **Topbot Publisher** - Publish stereo topbot images from disk to Hammerhead over ROS2

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

5.1.7 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 (`Tx < 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

5.1.8 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_;
};
```

5.1.9 DDS Transport Configuration

When sending or receiving large images (e.g. full-resolution topbot stereo pairs), the ROS2 middleware is almost always problematic. We have tried several different DDS suppliers, and our conclusion is that the default middleware is sufficient as long

as you tune it for high-throughput image transfer. The easiest way we have found to do this is to use the XML profiles we include in the `config/` folder.

TLDR

In the terminal where hammerhead runs AND the terminal where the sender/reciever runs, you should run this command first:

```
export FASTRTPS_DEFAULT_PROFILES_FILE=/path/to/config/fastdds.xml
```

Detailed Explanation

1. If sending and receiving messages on the same machine, you should to use shared memory, not the networking stack. The default shared memory segment seems to be tuned for fast, small messages. The aforementioned XML increases it by orders of magnitude.
2. If you are sending over the network, the issue is also the buffer size. The default size is small, meaning that large images are broken into many small packets, and dropping a single one of those packets then brings the whole pipeline to a halt. The aforementioned XML increases the buffer size so that messages are sent in fewer, larger packets.

The `config/` folder contains 2 XML profiles for tuning DDS:

- `fastdds.xml` : Tells Ros2 (technically the FASTRTPS middleware) to use shared memory by default, and UDP as a fallback. Both are set up with large buffer sizes. The builtin transports (with small buffers) are explicitly disabled.
- `fastdds_udp.xml` : Tells Ros2 to only use UDP with large buffers.

Note: The 8 MB buffer sizes require the kernel to allow large socket buffers. If the defaults are too small (common on stock Linux), increase them:

```
# Check current limits
sysctl net.core.rmem_max net.core.wmem_max

# Set to 8 MB (must be >= the buffer sizes in the XML)
sudo sysctl -w net.core.rmem_max=8388608
sudo sysctl -w net.core.wmem_max=8388608
```

To make this persistent across reboots, add to `/etc/sysctl.conf` :

```
net.core.rmem_max=8388608
net.core.wmem_max=8388608
```

Then apply immediately with:

```
sudo sysctl -p
```

Without this, the kernel silently caps the buffer size and large image transfers will be slow.

Troubleshooting

If you feel like Hammerhead is running slow, follow these steps:

1. Check that you're using FastDDS (not CycloneDDS or another middleware):

```
echo $RMW_IMPLEMENTATION
```

This should be empty (FastDDS is the default) or `rmw_fastrtps_cpp` . If it's set to something else (e.g. `rmw_cyclonedds_cpp`), the XML profiles will be silently ignored. Unset it:

```
unset RMW_IMPLEMENTATION
```

1. Check that the XML profile is being loaded:

When a ROS2 node starts, FastDDS will print XML parsing errors to stderr if the file is malformed or missing. If you see no errors and no improvement, the file is likely not being read. Verify:

```
echo $FASTRTPS_DEFAULT_PROFILES_FILE
```

If publisher and subscriber are on the same machine, then you can check that shared memory is actually used with something like:

```
ls /dev/shm/ | grep fast
```

If you see `fastrtps_*` files, SHM is active. If not, FastDDS fell back to UDP - usually because the segment size is too small for your messages.

1. If you are unsure of whether these profiles are having an effect, we recommend running hammerhead and the publisher subscriber 3 times...
2. In the first run, try with the Ros2 defaults in the terminal where Hammerhead and the publisher and/or subscriber are running:

```
unset FASTRTPS_DEFAULT_PROFILES_FILE
```

That is the performance you would get without tuning the DDS.

- Next, you can see what the throughput would be if Ros2 could only use UDP with large buffers by running this in all of your terminals:

```
export FASTRTPS_DEFAULT_PROFILES_FILE=$(pwd)/config/fastdds_udp.xml
```

- Finally, you can see what type of performance you can achieve when Ros2 is allowed to use shared memory for communication:

```
export FASTRTPS_DEFAULT_PROFILES_FILE=$(pwd)/config/fastdds.xml
```

We suspect that the default configs we supply will drastically increase your throughput compared to the defaults.

Important Notes

- The environment variable must be set in **every terminal** that runs a ROS2 node (both publisher and subscriber). If only one side is configured, they will still communicate (both have UDP in common), but the unconfigured side becomes the bottleneck — its default buffer sizes (~512KB for SHM, ~212KB for UDP) are far too small for large images, and you'll see the same slow/dropped frame behavior as having no config at all.
- The publisher and subscriber should use **compatible transports**. The SHM config includes UDP as a fallback, so it is compatible with the UDP-only config. However, you won't get SHM benefits unless both sides have it enabled.

5.1.10 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

5.2 Python Examples

5.2.1 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

5.2.2 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
`/nodar/left/image_raw`	Raw left camera
`/nodar/right/image_raw`	Raw right camera
`/nodar/left/image_rect`	Rectified left image
`/nodar/right/image_rect`	Rectified right image
`/nodar/disparity`	Disparity map
`/nodar/color_blended_depth/image_raw`	Color-coded depth visualization
`/nodar/topbot_raw`	Raw topbot image (left is top-half, right is bottom-half)
`/nodar/topbot_rect`	Rectified topbot image (left is top-half, right is bottom-half)

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 then 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.

5.2.3 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

5.2.4 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

See the DDS Transport Configuration section in the main README for how to configure shared memory or tuned UDP transport.

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

5.2.5 Topbot Publisher

Hammerhead can consume stereo images from Lucid cameras, over ZMQ, or over ROS2. This example publishes vertically stacked stereo images (which we refer to as `topbot` images) from disk onto a ROS2 topic so that Hammerhead can consume them as a camera source.

Hammerhead Configuration

To use this publisher with Hammerhead, set the camera source in `master_config.ini`:

```
camera_source = ros2:///nodar/topbots
```

The topic in the config must match the `--topic` argument (default: `/nodar/topbots`).

Build

```
cd ros2/nodarhub
colcon build --packages-up-to topbot_publisher_py
source install/setup.bash
```

Usage

```
ros2 run topbot_publisher_py topbot_publisher_py <topbot_data_directory> [--topic /nodar/topbots] [--fps 10] [--encoding auto]
```

PARAMETERS

- `image_dir`: Path to directory containing topbot images (default: `.`)
- `--topic`: ROS2 topic name (default: `/nodar/topbots`)
- `--fps`: Publish rate in frames per second (default: `10`)
- `--encoding`: Image encoding: `auto`, `mono8`, `mono16`, `bgr8`, `bgr16`, `bayer_bggr8`, `bayer_rggb8`, etc. (default: `auto`)
- `--loop`: Loop over the images continuously (default: off)

EXAMPLES

```
# Publish topbot images with auto-detected encoding at 10 fps
ros2 run topbot_publisher_py topbot_publisher_py /path/to/topbot/data

# Publish at 5 fps on a custom topic
ros2 run topbot_publisher_py topbot_publisher_py /path/to/topbot/data --topic /camera/image_raw --fps 5

# Publish with explicit Bayer encoding, looping continuously
ros2 run topbot_publisher_py topbot_publisher_py /path/to/topbot/data --encoding bayer_rggb8 --loop
```

Supported Image Formats

- TIFF (`.tiff`, `.tif`)
- PNG (`.png`)

Features

- Publish pre-recorded stereo image pairs to Hammerhead over ROS2
- Automatic encoding detection from image properties
- Configurable frame rate and topic name
- Single-pass playback by default; continuous looping with `--loop`
- Per-frame timing output for performance monitoring

Requirements

- ROS2 with `rclpy`, `sensor_msgs`, and `cv_bridge`
- OpenCV (`cv2`)
- Directory should contain topbot image files in a supported format

DDS Transport

For large images, the default DDS transport may be too slow. See the DDS Transport Configuration section in the main README for how to configure shared memory or tuned UDP transport.

Troubleshooting

- **Images not found**: Ensure the directory contains TIFF or PNG files
- **Encoding errors**: Try specifying `--encoding` explicitly instead of `auto`
- **Slow publish rate**: Large images may exceed the requested period. Check the DDS Transport Configuration — switching to shared memory transport typically resolves this

Press `Ctrl+C` to stop publishing.

5.2.6 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

5.3 C++ Examples

5.3.1 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

5.3.2 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
`/nodar/left/image_raw`	Raw left camera
`/nodar/right/image_raw`	Raw right camera
`/nodar/left/image_rect`	Rectified left image
`/nodar/right/image_rect`	Rectified right image
`/nodar/disparity`	Disparity map
`/nodar/color_blended_depth/image_raw`	Color-coded depth visualization
`/nodar/topbot_raw`	Raw topbot image (left is top-half, right is bottom-half)
`/nodar/topbot_rect`	Rectified topbot image (left is top-half, right is bottom-half)

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 then 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.

5.3.3 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

5.3.4 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

See the DDS Transport Configuration section in the main README for how to configure shared memory or tuned UDP transport.

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

5.3.5 Topbot Publisher

Hammerhead can consume stereo images from Lucid cameras, over ZMQ, or over ROS2. This example publishes vertically stacked stereo images (which we refer to as `topbot` images) from disk onto a ROS2 topic so that Hammerhead can consume them as a camera source.

Hammerhead Configuration

To use this publisher with Hammerhead, set the camera source in `master_config.ini`:

```
camera_source = ros2:///nodar/topbots
```

The topic in the config must match the `--topic` argument (default: `/nodar/topbots`).

Build

```
cd ros2/nodarhub
colcon build --packages-up-to topbot_publisher
source install/setup.bash
```

Usage

```
ros2 run topbot_publisher topbot_publisher <topbot_data_directory> [--topic /nodar/topbots] [--fps 10] [--encoding auto]
```

PARAMETERS

- `image_dir`: Path to directory containing topbot images (default: `.`)
- `--topic`: ROS2 topic name (default: `/nodar/topbots`)
- `--fps`: Publish rate in frames per second (default: `10`)
- `--encoding`: Image encoding: `auto`, `mono8`, `mono16`, `bgr8`, `bgr16`, `bayer_bggr8`, `bayer_rggb8`, etc. (default: `auto`)
- `--loop`: Loop over the images continuously (default: off)

EXAMPLES

```
# Publish topbot images with auto-detected encoding at 10 fps
ros2 run topbot_publisher topbot_publisher /path/to/topbot/data

# Publish at 5 fps on a custom topic
ros2 run topbot_publisher topbot_publisher /path/to/topbot/data --topic /camera/image_raw --fps 5

# Publish with explicit Bayer encoding, looping continuously
ros2 run topbot_publisher topbot_publisher /path/to/topbot/data --encoding bayer_rggb8 --loop
```

Supported Image Formats

- TIFF (`.tiff`, `.tif`)
- PNG (`.png`)

Features

- Publish pre-recorded stereo image pairs to Hammerhead over ROS2
- Automatic encoding detection from image properties
- Configurable frame rate and topic name
- Single-pass playback by default; continuous looping with `--loop`
- Per-frame timing output for performance monitoring

Requirements

- ROS2 with `rclcpp`, `sensor_msgs`, and `cv_bridge`
- OpenCV (imgcodecs)
- Directory should contain topbot image files in a supported format

Alternative Usage

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

```
mkdir build && cd build
cmake .. && make
./topbot_publisher /path/to/topbot/data
```

DDS Transport

For large images, the default DDS transport may be too slow. See the DDS Transport Configuration section in the main README for how to configure shared memory or tuned UDP transport.

Troubleshooting

- **Images not found**: Ensure the directory contains TIFF or PNG files
- **Encoding errors**: Try specifying `--encoding` explicitly instead of `auto`
- **Slow publish rate**: Large images may exceed the requested period. Check the DDS Transport Configuration — switching to shared memory transport typically resolves this

Press `Ctrl+C` to stop publishing.

5.3.6 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

6. ZMQ

6.1 Hammerhead ZMQ

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

6.1.1 Table of Contents

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

6.1.2 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

6.1.3 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 enviroment 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/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/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/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
```

6.1.4 Message Types & Ports

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

Image Streams

Port	Topic	Description	Message Type
9800	`nodar/left/image_raw`	Raw left camera feed	`StampedImage`
9801	`nodar/right/image_raw`	Raw right camera feed	`StampedImage`
9802	`nodar/left/image_rect`	Rectified left image	`StampedImage`
9803	`nodar/right/image_rect`	Rectified right image	`StampedImage`
9804	`nodar/disparity`	Disparity map (Q12.4 format)	`StampedImage`
9805	`nodar/color_blended_depth/image_raw`	Color-coded depth visualization	`StampedImage`
9813	`nodar/topbot_raw`	Raw top (left) and bottom (right) camera pair	`StampedImage`
9823	`nodar/topbot_rect`	Rectified top (left) and bottom (right) camera pair	`StampedImage`
9815	`nodar/confidence_map`	Confidence map	`StampedImage`
9900	`nodar/occupancy_map`	Occupancy map	`StampedImage`

3D Data Streams

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

Detection & Control

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

6.1.5 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.

6.1.6 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
- **Topbot Metadata Writer** - Write details YAML metadata into topbot TIFF Software tags

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

6.1.7 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 (`Tx < 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

6.1.8 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

6.1.9 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

6.2 Python Examples

6.2.1 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
`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
`nodar/confidence_map`	9815	Confidence map
`nodar/occupancy_map`	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.

6.2.2 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
`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
`nodar/confidence_map`	9815	Confidence map

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.

6.2.3 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.

6.2.4 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.

6.2.5 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.

6.2.6 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
`nodar/occupancy_map`	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.

6.2.7 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.

6.2.8 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.

6.2.9 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.

6.2.10 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.

6.2.11 Topbot Metadata Writer

Write details YAML metadata into the TIFF Software tag (305) of topbot TIFF files.

Installation

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

Usage

```
topbot_metadata_writer <folder>
```

PARAMETERS

- `folder` : Path to a folder containing `topbot/` and `details/` subdirectories

EXPECTED FOLDER STRUCTURE

```
folder/
  topbot/
    000000.tiff
    000001.tiff
    ...
  details/
    000000.yaml
    000001.yaml
    ...
```

Each YAML file in `details/` is matched to the TIFF file in `topbot/` with the same stem name. TIFF files without a matching YAML are skipped.

EXAMPLE

```
topbot_metadata_writer /path/to/recording
```

How it works

For each TIFF file in `topbot/`, the tool:

1. Reads the matching YAML file from `details/`
2. Formats the YAML data into a `DETAILS: "..."` string
3. Injects it as the TIFF Software tag (305) in every IFD
4. Writes the updated TIFF back in place

6.2.12 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 ≥90°C, Error ≥99°C
- **GPU Temperature**: Warning ≥90°C, Error ≥99°C
- **GPU Utilization**: Warning ≥99.0%, Error ≥99.5%
- **GPU Memory**: Warning ≥80%, Error ≥90%
- **RAM Usage**: Warning ≥80%, Error ≥90%
- **CPU Load**: Warning ≥11.0, Error ≥20.0 (load average)
- **Disk Space**: Warning ≤10GB free per mount point

HAMMERHEAD MONITORING

- **Frame Sync**: Warning ≥50µs, Error ≥1000µs (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.

6.2.13 Navigation Publisher

Publish navigation data (IMU, GPS, Odometry) with coordinate system transformations to Hammerhead.

Installation

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

Configuration

Before running the navigation publisher, ensure the following settings are configured in `master_config.ini`:

```
# [string] Navigation publisher source.
# Specify the IP address of the navigation publisher (e.g., 127.0.0.1).
# Leave empty to disable navigation data.
navigation_publisher = 127.0.0.1
```

Note: Set `navigation_publisher` to the IP address where the navigation publisher is running. Use `127.0.0.1` for local testing.

EXAMPLES

```
# Run the navigation publisher
python navigation_publisher
```

The publisher will continuously send navigation data at 10 Hz with example values:

- **IMU**: Acceleration (0, 0, 9.81) m/s², Gyro (0, 0, 0) rad/s, Magnetometer (0, 0, 0) gauss, Temperature 25°C
- **GPS**: Lat 0.0°, Lon 0.0°, Alt 10.0m, Speed 5.0 m/s, Fix type 0, 1 satellite
- **Odometry**: Position (0, 0, 0)m, Velocity (5, 0, 0) m/s, Angular velocity (0, 0, 0) rad/s

Features

- Python implementation for easy integration and customization
- Publishes comprehensive navigation data including IMU, GPS, and Odometry
- Individual timestamps for each sensor (IMU, GPS, Odometry)
- Includes 4x4 transformation matrix from body frame to Nodar raw camera frame
- ZMQ-based communication on port 9824 (topic: `nodar/navigation`)
- Publishes at 10 Hz update rate

Coordinate Systems

The body/odometry frame used in this example is:

BODY/ODOMETRY FRAME (EXAMPLE)

- **x**: Forward direction
- **y**: Left direction
- **z**: Up direction

Note: You can use any body frame coordinate system. Simply adjust the `T_body_to_raw_camera` transformation matrix accordingly to match your coordinate system.

NODAR RAW CAMERA FRAME

- **x**: Right direction
- **y**: Down direction
- **z**: Forward direction

Message Format

The NavigationData message contains:

- **Timestamp**: 64-bit nanosecond timestamp (message creation time)
- **IMU Data**:
- Timestamp (ns)
- Acceleration (x, y, z) in m/s²
- Gyroscope (x, y, z) in rad/s
- Magnetometer (x, y, z) in gauss
- Temperature in °C
- **GPS Data**:
- Timestamp (ns)
- Latitude, Longitude (degrees), Altitude (m)
- Horizontal/Vertical uncertainty (m)
- Speed (m/s), Course (degrees)
- Fix type, Number of satellites
- **Odometry Data**:
- Timestamp (ns)
- Position (x, y, z) in m
- Velocity (x, y, z) in m/s
- Angular velocity (x, y, z) in rad/s
- **Transformation Matrix**: 4x4 matrix from body to raw camera frame (row-major)

Use Cases

This example is useful for:

- Testing navigation data integration in perception systems
- Validating coordinate system transformations
- Providing multi-sensor navigation data (IMU, GPS, Odometry) to Hammerhead
- Sensor fusion and localization testing

Customization

To modify the navigation values, edit the field assignments in `navigation_publisher.py`:

```
# IMU data (example values)
nav_data.imu_acceleration_z_m_s2 = 9.81  # Gravity

# GPS data (example values)
nav_data.gps_altitude_m = 10.0

# Odometry data (example values)
nav_data.odom_velocity_x_m_s = 5.0  # 5 m/s forward
```

To change the transformation matrix, modify the `T_body_to_raw_camera` list accordingly.

Troubleshooting

- **Port conflict**: Ensure port 9824 is not in use by another application
- **Hammerhead not receiving data**: Verify `navigation_publisher` is set to the correct IP address in `master_config.ini`

Press `Ctrl+C` to stop publishing.

6.3 C++ Examples

6.3.1 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
`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
`nodar/confidence_map`	9815	Confidence map
`nodar/occupancy_map`	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.

6.3.2 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
`nodar/confidence_map`	9815	Confidence map

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.

6.3.3 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.

6.3.4 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.

6.3.5 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_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

- 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.

6.3.6 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
`nodar/occupancy_map`	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.

6.3.7 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.

6.3.8 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.

6.3.9 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.

6.3.10 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.

6.3.11 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_GRBG`
- `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.

6.3.12 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.

6.3.13 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 ≥90°C, Error ≥99°C
- **GPU Temperature**: Warning ≥90°C, Error ≥99°C
- **RAM Usage**: Warning ≥80%, Error ≥90%
- **CPU Load**: Warning ≥11.0, Error ≥20.0 (load average)
- **Disk Space**: Warning ≤10GB free per mount point

HAMMERHEAD MONITORING

- **Frame Sync**: Warning ≥50µs, Error ≥1000µs (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.

6.3.14 Navigation Publisher

Publish navigation data (IMU, GPS, Odometry) with coordinate system transformations to Hammerhead.

Build

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

Configuration

Before running the navigation publisher, ensure the following settings are configured in `master_config.ini`:

```
# [string] Navigation publisher source.
# Specify the IP address of the navigation publisher (e.g., 127.0.0.1).
# Leave empty to disable navigation data.
navigation_publisher = 127.0.0.1
```

Note: Set `navigation_publisher` to the IP address where the navigation publisher is running. Use `127.0.0.1` for local testing.

Usage

```
# Linux
./navigation_publisher

# Windows
./Release/navigation_publisher.exe
```

EXAMPLES

```
# Run the navigation publisher
./navigation_publisher
```

The publisher will continuously send navigation data at 10 Hz with example values:

- **IMU**: Acceleration (0, 0, 9.81) m/s², Gyro (0, 0, 0) rad/s, Magnetometer (0, 0, 0) gauss, Temperature 25°C
- **GPS**: Lat 0.0°, Lon 0.0°, Alt 10.0m, Speed 5.0 m/s, Fix type 0, 1 satellite
- **Odometry**: Position (0, 0, 0)m, Velocity (5, 0, 0) m/s, Angular velocity (0, 0, 0) rad/s

Features

- High-performance C++ implementation for real-time navigation data publishing
- Publishes comprehensive navigation data including IMU, GPS, and Odometry
- Individual timestamps for each sensor (IMU, GPS, Odometry)
- Includes 4x4 transformation matrix from body frame to Nodar raw camera frame
- ZMQ-based communication on port 9824 (topic: `nodar/navigation`)
- Publishes at 10 Hz update rate

Coordinate Systems

The body/odometry frame used in this example is:

BODY/ODOMETRY FRAME (EXAMPLE)

- **x**: Forward direction
- **y**: Left direction
- **z**: Up direction

Note: You can use any body frame coordinate system. Simply adjust the `T_body_to_raw_camera` transformation matrix accordingly to match your coordinate system.

NODAR RAW CAMERA FRAME

- **x**: Right direction
- **y**: Down direction
- **z**: Forward direction

Message Format

The NavigationData message contains:

- **Timestamp**: 64-bit nanosecond timestamp (message creation time)
- **IMU Data**:
- Timestamp (ns)
- Acceleration (x, y, z) in m/s²
- Gyroscope (x, y, z) in rad/s
- Magnetometer (x, y, z) in gauss
- Temperature in °C
- **GPS Data**:
- Timestamp (ns)
- Latitude, Longitude (degrees), Altitude (m)
- Horizontal/Vertical uncertainty (m)
- Speed (m/s), Course (degrees)
- Fix type, Number of satellites
- **Odometry Data**:
- Timestamp (ns)
- Position (x, y, z) in m
- Velocity (x, y, z) in m/s
- Angular velocity (x, y, z) in rad/s
- **Transformation Matrix**: 4x4 matrix from body to raw camera frame (row-major)

Use Cases

This example is useful for:

- Testing navigation data integration in perception systems
- Validating coordinate system transformations
- Providing multi-sensor navigation data (IMU, GPS, Odometry) to Hammerhead

Troubleshooting

- **Port conflict**: Ensure port 9824 is not in use by another application
- **Hammerhead not receiving data**: Verify `navigation_publisher` is set to the correct IP address in `master_config.ini`

Press `Ctrl+C` to stop publishing.

7. 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 `euler_x_deg` in the `stereo_sensor_coordinate.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.