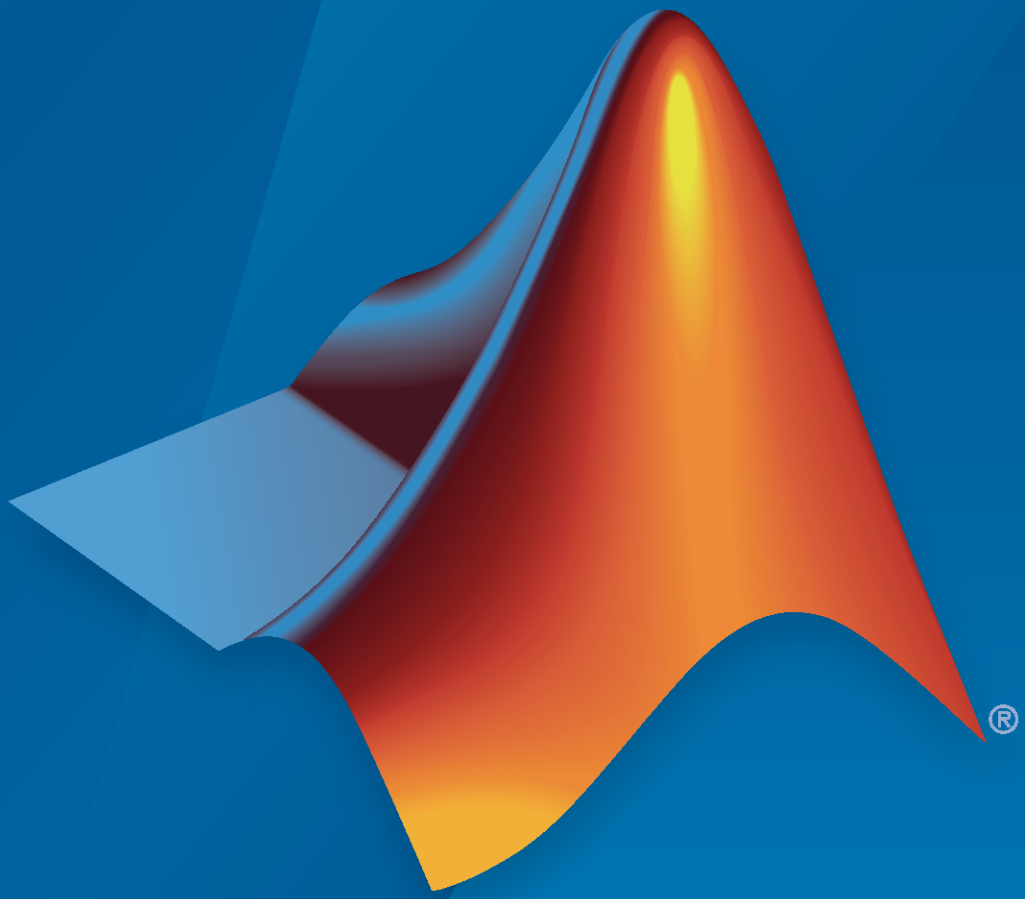
Lidar Toolbox™

Getting Started Guide



MATLAB®

R2023a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Lidar Toolbox™ Getting Started Guide

Revision History

September 2020	Online only	New for Version 1.0 (R2020b)
March 2021	Online only	Revised for Version 1.1 (R2021a)
September 2021	Online only	Revised for Version 2.0 (R2021b)
March 2022	Online only	Revised for Version 2.1 (R2022a)
September 2022	Online only	Revised for Version 2.2 (R2022b)
March 2023	Online only	Revised for Version 2.3 (R2023a)

Contents

1 Introduction to Lidar Toolbox

2 Concept Pages

Introduction to Lidar Toolbox

Lidar Toolbox Product Description

Design, analyze, and test lidar processing systems

Lidar Toolbox™ provides algorithms, functions, and apps for designing, analyzing, and testing lidar processing systems. You can perform object detection and tracking, semantic segmentation, shape fitting, lidar registration, and obstacle detection. The toolbox provides workflows and an app for lidar-camera cross-calibration.

The toolbox lets you stream data from Velodyne® lidars and read data recorded by Velodyne and IBEO lidar sensors. The Lidar Viewer App enables interactive visualization and analysis of lidar point clouds. You can train detection, semantic segmentation, and classification models using machine learning and deep learning algorithms such as PointPillars, SqueezeSegV2, and PointNet++. The Lidar Labeler App supports manual and semi-automated labeling of lidar point clouds for training deep learning and machine learning models.

Lidar Toolbox provides lidar processing reference examples for perception and navigation workflows. Most toolbox algorithms support C/C++ code generation for integrating with existing code, desktop prototyping, and deployment.

Concept Pages

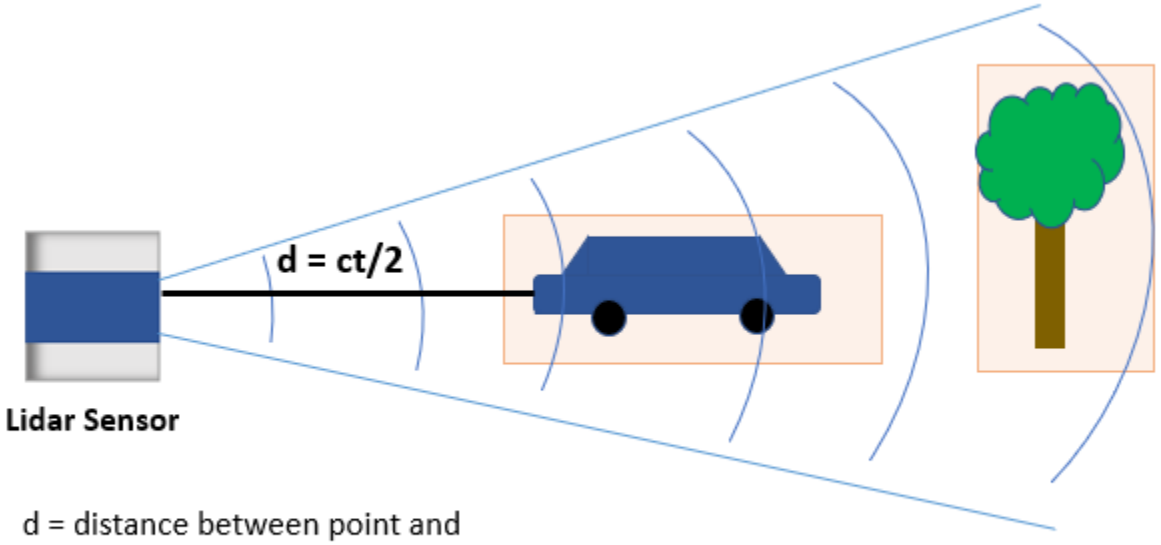
Introduction to Lidar Processing

What is Lidar?

Lidar, which stands for Light Detection and Ranging, is a method of 3-D laser scanning.

Lidar sensors provide 3-D structural information about an environment. Advanced driving assistance systems (ADAS), robots, and unmanned aerial vehicles (UAVs) employ lidar sensors for accurate 3-D perception, navigation, and mapping.

Lidar is an active remote sensing system that uses laser light to measure the distance of the sensor from objects in a scene. A lidar sensor emits laser pulses that reflect off of surrounding objects. The sensor then captures this reflected light and uses the time-of-flight principle to measure its distance from objects, enabling it to perceive the structure of its surroundings.



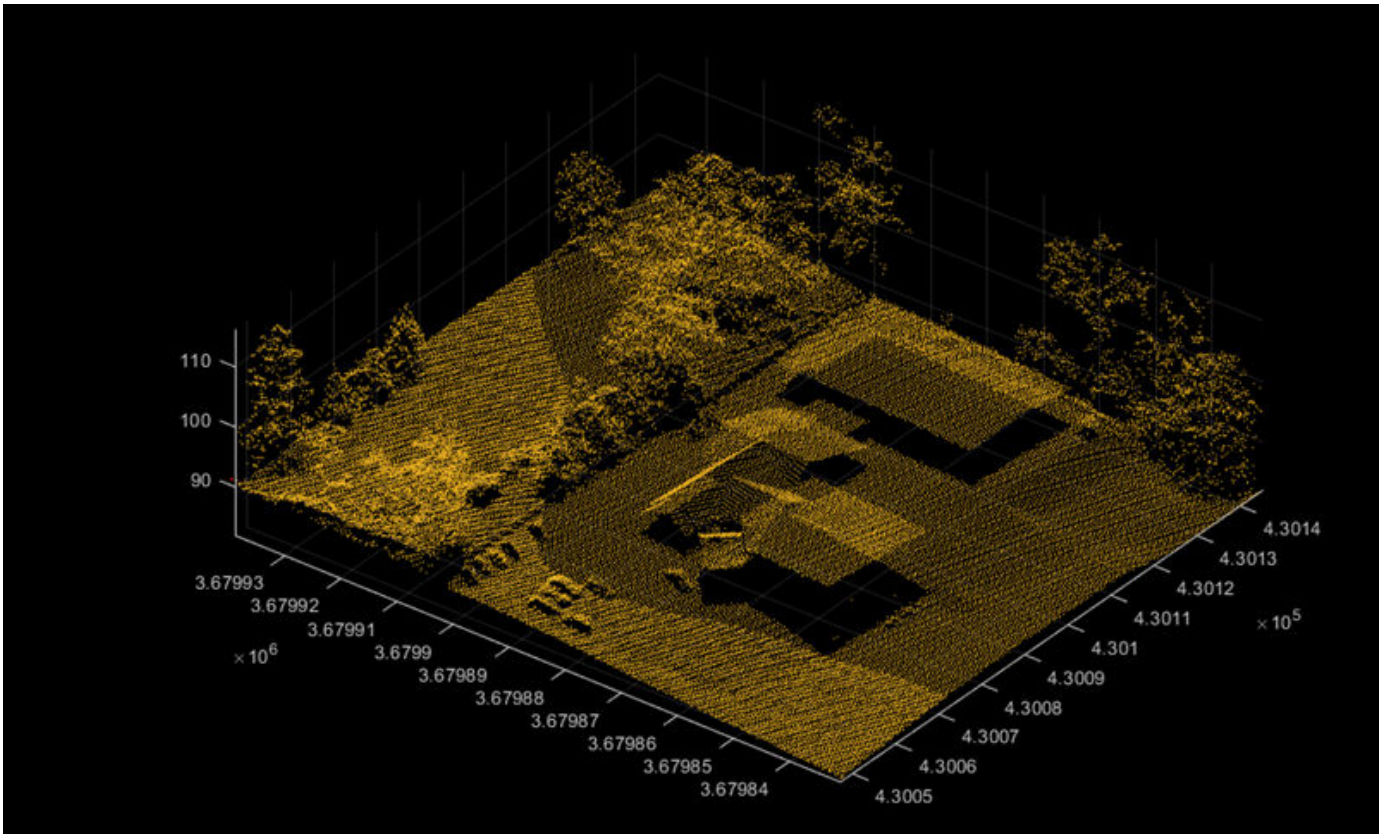
d = distance between point and the sensor
c = speed of light
t = time of flight

A lidar sensor stores these reflected laser pulses, or laser returns, as a collection of points. This collection of points is called a point cloud.

What is a Point Cloud?

A point cloud is a collection of 3-D points in space. Just as an image is the output of a camera, a point cloud is the output of a lidar sensor.

A lidar sensor captures attributes such as the location in *xyz*-coordinates, the intensity of the laser light, and the surface normal at each point of a point cloud. With this information a point cloud generates a 3-D map of an environment. You can store and process the information from a point cloud in MATLAB® by using a `pointCloud` object.



Point clouds can be either unorganized or organized. In an unorganized point cloud, the points are stored as a single stream of 3-D coordinates. In an organized point cloud, the points are arranged into rows and columns based on spatial relation between them. For more information about organized and unorganized point clouds, see “What are Organized and Unorganized Point Clouds?”

Types of Lidar

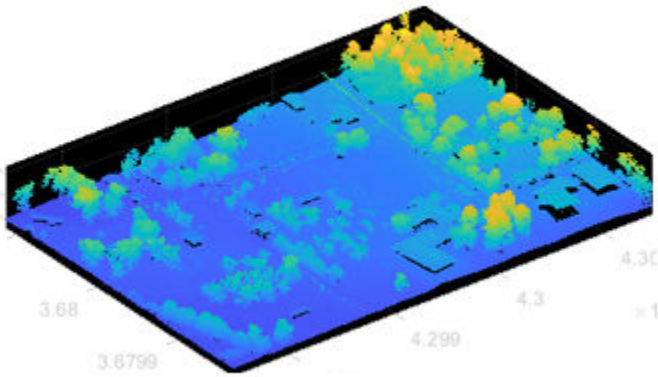
You can broadly divide the various types of lidar sensors based on whether they are for airborne or terrestrial application.

Airborne or Aerial Lidar

Airborne lidar sensors are those attached to helicopters, aircrafts, or UAVs. They consist of topographic and bathymetric sensors.

- Topographic sensors help in monitoring and mapping the topography of a region. Applications include urban planning, landscape ecology, forest planning and mapping.
- Bathymetric sensors estimate the depth of water bodies. These sensors have an additional green laser that travels through a water column. Applications include coastline management, and oceanography.

Aerial lidar sensor

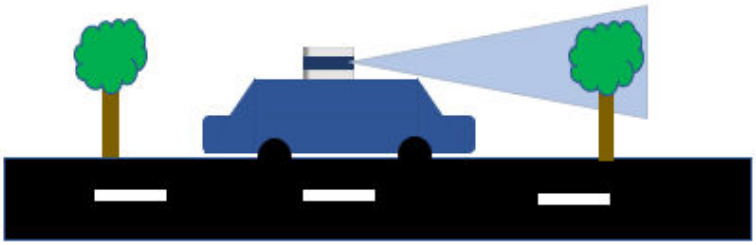


Aerial point cloud data

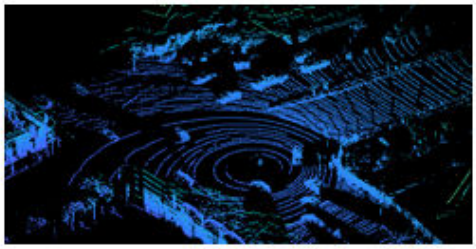
Terrestrial Lidar

Terrestrial lidar sensors scan the surface of the Earth or the immediate surroundings of the sensor on land. These sensors can be static or mobile.

- Static sensors collect point clouds from a fixed location. Applications such as mining, archaeology, smartphones, and architecture use static sensors.
- Mobile sensors are most commonly used in autonomous driving systems, and are mounted on vehicles. Other applications include robotics, transport planning, and mapping.



Terrestrial lidar sensor



Terrestrial point cloud data

Advantages of Lidar Technology

Lidar sensors are useful when you need to take accurate measurements at longer distances and higher resolutions than is possible with radar sensors, or in environmental or lighting conditions that would negatively affect a camera. Lidar scans are also natively 3-D, and do not require additional software to add depth.

You can use lidar sensors to detect small details, scan dense environments, and collect data at night or in inclement weather, all with high speed.

Lidar Processing Overview

I/O and Supported Hardware

Because the wide variety of lidar sensors available from companies such as Velodyne, Ouster®, Hesai®, and Ibeo® use a variety of formats for point cloud data, Lidar Toolbox provides tools to import and export point clouds using various file formats. Lidar Toolbox currently supports reading data from the PLY, PCAP, PCD, LAS, LAZ, and Ibeo data container (IDC) sensor formats. You can also write point cloud data to the PLY, PCD, and LAS formats. For more information on file input and output, see "I/O". You can also stream live data from Velodyne and Ouster sensors. For more details on streaming live data, see "Lidar Toolbox Supported Hardware".

Preprocessing

Lidar Toolbox enables you perform preprocessing operations such as downsampling, denoising, and cropping on your point cloud data. To learn more about visualizing and preprocessing point clouds, see "Get Started with Lidar Viewer".

Labeling, Segmentation and Detection

Labeling objects in a point cloud helps you organize and analyze ground truth data for object detection and segmentation. To learn more about lidar labeling, see "Get Started with the Lidar Labeler".

Many applications for lidar processing rely on deep learning algorithms to segment, detect, track, and analyze objects of interest in a point cloud. To learn more about point cloud processing using deep learning, see "Getting Started with Point Clouds Using Deep Learning".

Calibration and Sensor Fusion

Most modern sensing systems use sensor suites that contain multiple sensors. To obtain meaningful information from multiple sensors, you must first calibrate these sensors. Calibration is the process of aligning the coordinate systems of multiple sensors through rotational and translational transformations. For more information about coordinate systems, see "Coordinate Systems in Lidar Toolbox".

Lidar Toolbox provides various tools for calibration and sensor fusion. Many applications involve capturing the same scene using both a lidar sensor and a camera. To construct an accurate 3-D scene, you must fuse the data from these sensors by first calibrating them to one another. For more information on lidar-camera fusion, see "What Is Lidar-Camera Calibration?" and "Get Started with Lidar Camera Calibrator".

Navigation and Mapping

Mapping is the process of building a map of the environment around an autonomous system. You can use tools in Lidar Toolbox to perform simultaneous localization and mapping (SLAM), which is the process of calculating the position and orientation of the system, with respect to its surroundings, while simultaneously mapping its environment. For more information, see "Implement Point Cloud SLAM in MATLAB".

Applications of Lidar Technology

Lidar Toolbox provides many tools for typical workflows in different applications of lidar processing.

- **Autonomous Driving Assistance Systems** — You can detect cars, trucks, and other objects using the lidar sensors mounted on moving vehicles. You can semantically segment these point clouds to detect and track objects as they move. To learn more about vehicle detection and tracking using Lidar Toolbox, see the “Detect, Classify, and Track Vehicles Using Lidar” example.
- **Remote Sensing** — Airborne lidar sensors can generate point clouds that provide information about the vegetation cover in an area. To learn more about remote sensing using Lidar Toolbox, see the “Extract Forest Metrics and Individual Tree Attributes from Aerial Lidar Data” example.
- **Navigation and Mapping** — You can build a map using the lidar data generated from a vehicle-mounted lidar sensor. You can use these maps for localization and navigation. To learn more about map building, see the “Feature-Based Map Building from Lidar Data” example.

See Also

More About

- “What are Organized and Unorganized Point Clouds?”
- “Getting Started with Point Clouds Using Deep Learning”
- “Coordinate Systems in Lidar Toolbox”
- “What Is Lidar-Camera Calibration?”
- “Implement Point Cloud SLAM in MATLAB”

Coordinate Systems in Lidar Toolbox

A lidar sensor uses laser light to construct a 3-D scan of its environment. By emitting laser pulses into the surrounding environment and capturing the reflected pulses, the sensor can use the time-of-flight principle to measure its distance from objects in the environment. The sensor stores this information as a point cloud, which is a collection of 3-D points in space.

Lidar sensors record point cloud data with respect to either a local coordinate system, such as the coordinate system of the sensor or the ego vehicle, or in the world coordinate system.

World Coordinate System

The world coordinate system is a fixed universal frame of reference for all the vehicles, sensors, and objects in a scene. In a multisensor system, each sensor captures data in its own coordinate system. You can use the world coordinate system as a reference to transform data from different sensors into a single coordinate system.

Lidar Toolbox uses the right-handed Cartesian world coordinate system defined in ISO 8855, where the x-axis is positive in the direction of ego vehicle movement, the y-axis is positive to the left with regard to ego vehicle movement, and the z-axis is positive going up from the ground.

Sensor Coordinate System

A sensor coordinate system is one local to a specific sensor, such as a lidar sensor or a camera, with its origin located at the center of the sensor.

A lidar sensor typically measures the distance of objects from the sensor and collects the information as points. Each point has spherical coordinates of the form (r, Θ, Φ), which you can use to calculate the xyz-coordinates of a point.

- r is the distance of the point from the origin
- Φ is the azimuth angle in the XY-plane measured from the positive x-axis
- Θ is the elevation angle in the YZ-plane measured from positive z-axis

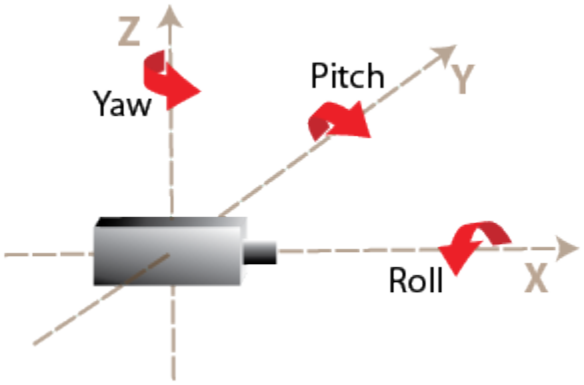
The mounting position and orientation of a lidar sensor can vary depending on its intended application.

Horizontal mounting used on ground vehicles for autonomous driving applications, enables the sensor to scan and detect objects and terrain in its immediate vicinity for applications such as obstacle avoidance.

Vertical mounting which is used on aerial vehicles, enables the sensor to scan large landscapes such as forests, plains, and bodies of water. Vertical mounting requires these additional parameters to define the sensor orientation.

- Roll — Angle of rotation around the front-to-back axis, which is the x-axis of the sensor coordinate system.
- Pitch — Angle of rotation around the side-to-side axis, which is the y-axis of the sensor coordinate system.

- Yaw — Angle of rotation around the vertical axis, which is the *z*-axis of the sensor coordinate system.



Coordinate System Transformation

Applications such as autonomous driving, mining, and underwater navigation use multisensor systems for a more complete understanding of the surrounding environment. Because each sensor captures data in its respective coordinate system, to fuse data from the different sensors in such a setup, you must define a reference coordinate frame and transform the data from all the sensors into reference frame coordinates.

For example, you can select the coordinate system of a lidar sensor as your reference frame. To create reference frame coordinates from your lidar sensor data, you must transform that data by multiplying it with the extrinsic parameters of the lidar sensor.

To apply transformation, you must multiply your data with the extrinsic parameters of the lidar sensor. The extrinsic parameter matrix consists of the translation vector and rotation matrix. The translation vector translates the origin of your selected reference frame to the origin of the world frame, and the rotation matrix rotates the axes to the correct orientation.

Lidar sensors and cameras are often used together to generate accurate 3-D scans of a scene. For more information about lidar-camera data fusion, see “What Is Lidar-Camera Calibration?”

See Also

More About

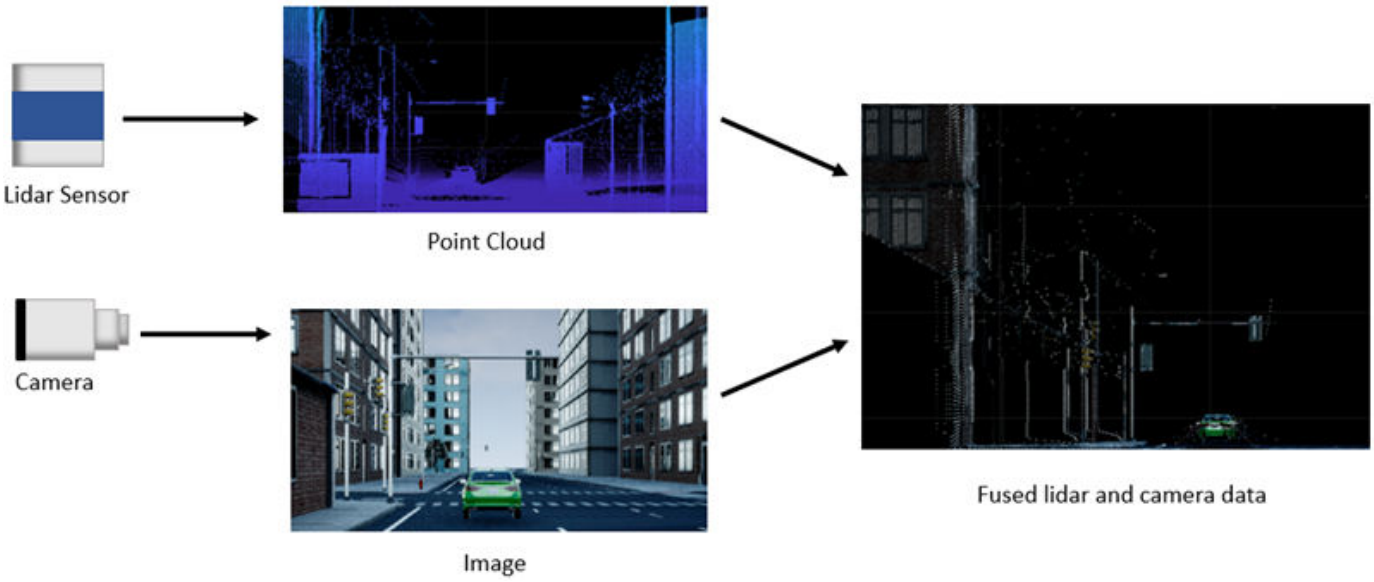
- “Coordinate Systems”
- “Image Coordinate Systems”

What Is Lidar-Camera Calibration?

Lidar-camera calibration establishes correspondences between 3-D lidar points and 2-D camera data to fuse the lidar and camera outputs together.

Lidar sensors and cameras are widely used together for 3-D scene reconstruction in applications such as autonomous driving, robotics, and navigation. While a lidar sensor captures the 3-D structural information of an environment, a camera captures the color, texture, and appearance information. The lidar sensor and camera each capture data with respect to their own coordinate system.

Lidar-camera calibration consists of converting the data from a lidar sensor and a camera into the same coordinate system. This enables you to fuse the data from both sensors and accurately identify objects in a scene. This figure shows the fused data.



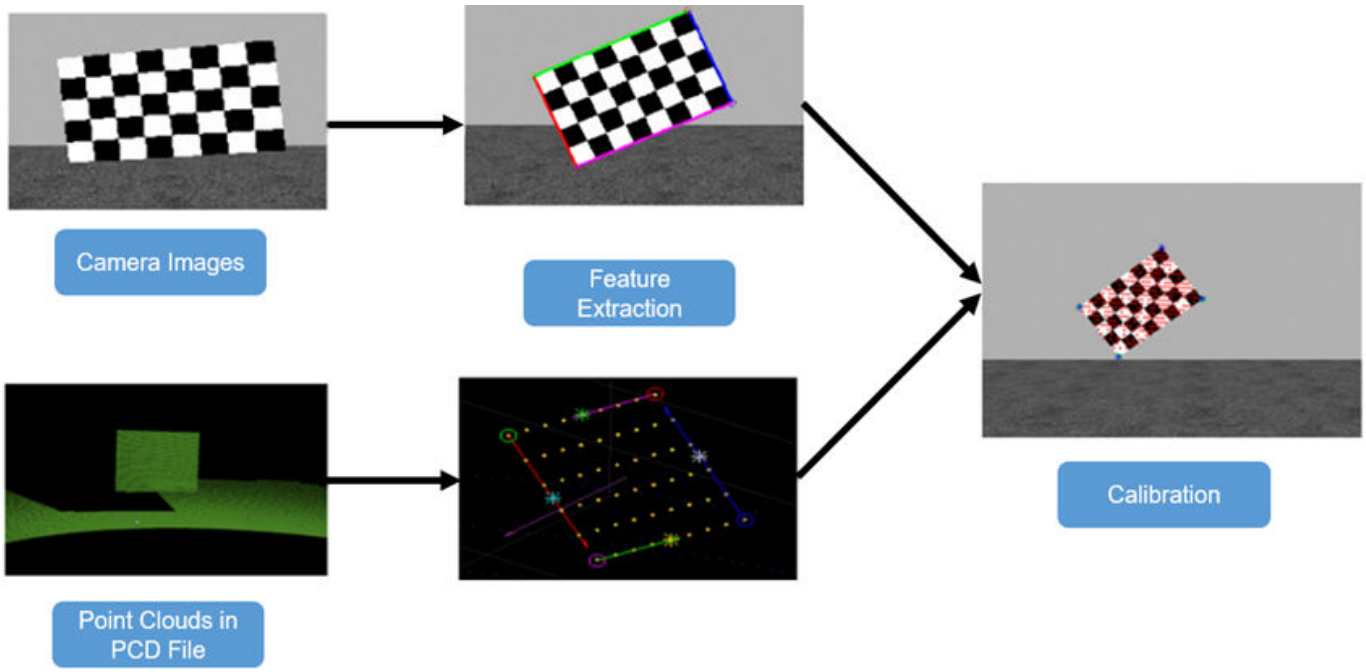
Lidar-camera calibration consists of intrinsic calibration and extrinsic calibration.

- Intrinsic calibration — Estimate the internal parameters of the lidar sensor and camera.
  - Manufacturers calibrate the intrinsic parameters of their lidar sensors in advance.
  - You can use the `estimateCameraParameters` function to estimate the intrinsic parameters of the camera, such as focal length, lens distortion, and skew. For more information, see the “Single Camera Calibration” example.
  - You can also interactively estimate camera parameters using the **Camera Calibrator** app.
- Extrinsic calibration — Estimate the external parameters of the lidar sensor and camera, such as location, orientation, to establish relative rotation and translation between the sensors.

Extrinsic Calibration of Lidar and Camera

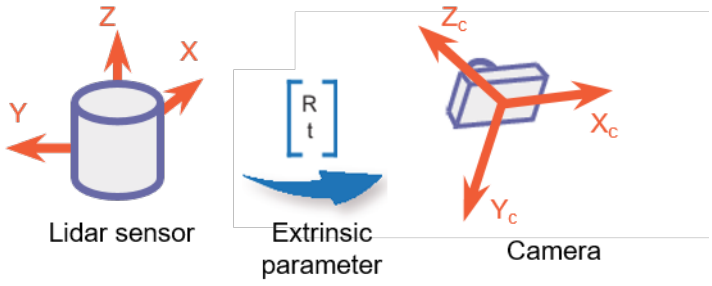
The extrinsic calibration of a lidar sensor and camera estimates a rigid transformation between them that establishes a geometric relationship between their coordinate systems. This process uses standard calibration objects, such as planar boards with checkerboard patterns.

This diagram shows the extrinsic calibration process for a lidar sensor and camera using a checkerboard.



The programmatic workflow for extrinsic calibration consists of these steps. Alternatively, you can use the **Lidar Camera Calibrator** app to interactively perform lidar-camera calibration.

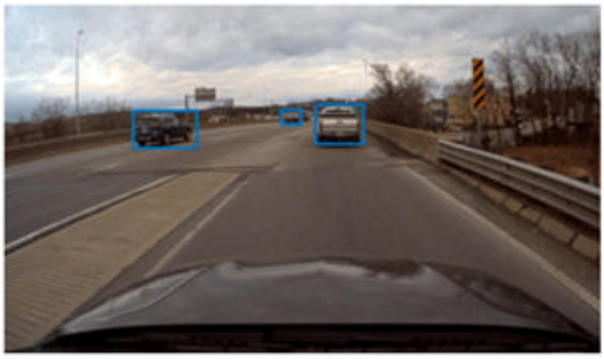
1. Extract the 3-D information of the checkerboard from both the camera and lidar sensor.
   a. To extract the 3-D checkerboard corners from the camera data, in world coordinates, use the `estimateCheckerboardCorners3d` function.
   b. To extract the checkerboard plane from the lidar point cloud data, use the `detectRectangularPlanePoints` function.
2. Use the checkerboard corners and planes to obtain the rigid transformation matrix, which consists of the rotation *R* and translation *t*. You can estimate the rigid transformation matrix by using the `estimateLidarCameraTransform` function. The function returns the transformation as a `rigidtform3d` object.



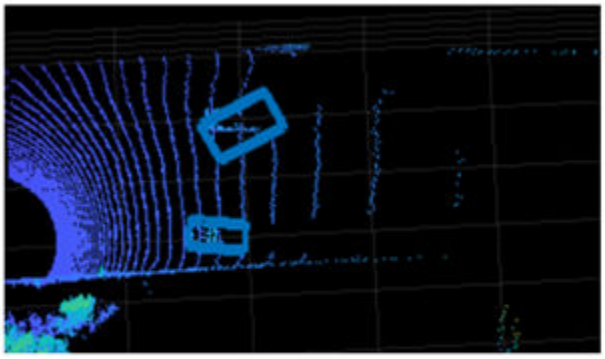
You can use the transformation matrix to:

- Evaluate the accuracy of your calibration by calculating the error. You can do so either programmatically, using `estimateLidarCameraTransform`, or interactively, using the **Lidar Camera Calibrator** app.
- Project lidar points onto an image by using the `projectLidarPointsOnImage` function, as shown in this figure.

- Fuse the lidar and camera outputs by using the `fuseCameraToLidar` function.
- Estimate the 3-D bounding boxes in a point cloud based on the 2-D bounding boxes in the corresponding image. For more information, see “Detect Vehicles in Lidar Using Image Labels”.



Bounding boxes in image



Detected bounding boxes in point cloud
(Top view)

References

[1] Zhou, Lipu, Zimo Li, and Michael Kaess. “Automatic Extrinsic Calibration of a Camera and a 3D LiDAR Using Line and Plane Correspondences.” In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5562–69. Madrid: IEEE, 2018. https://doi.org/10.1109/IROS.2018.8593660.

See Also

`estimateLidarCameraTransform` | `estimateCheckerboardCorners3d` | `detectRectangularPlanePoints` | `projectLidarPointsOnImage` | `fuseCameraToLidar` | `bboxCameraToLidar`

Related Examples

- “Lidar and Camera Calibration”
- “Detect Vehicles in Lidar Using Image Labels”

More About

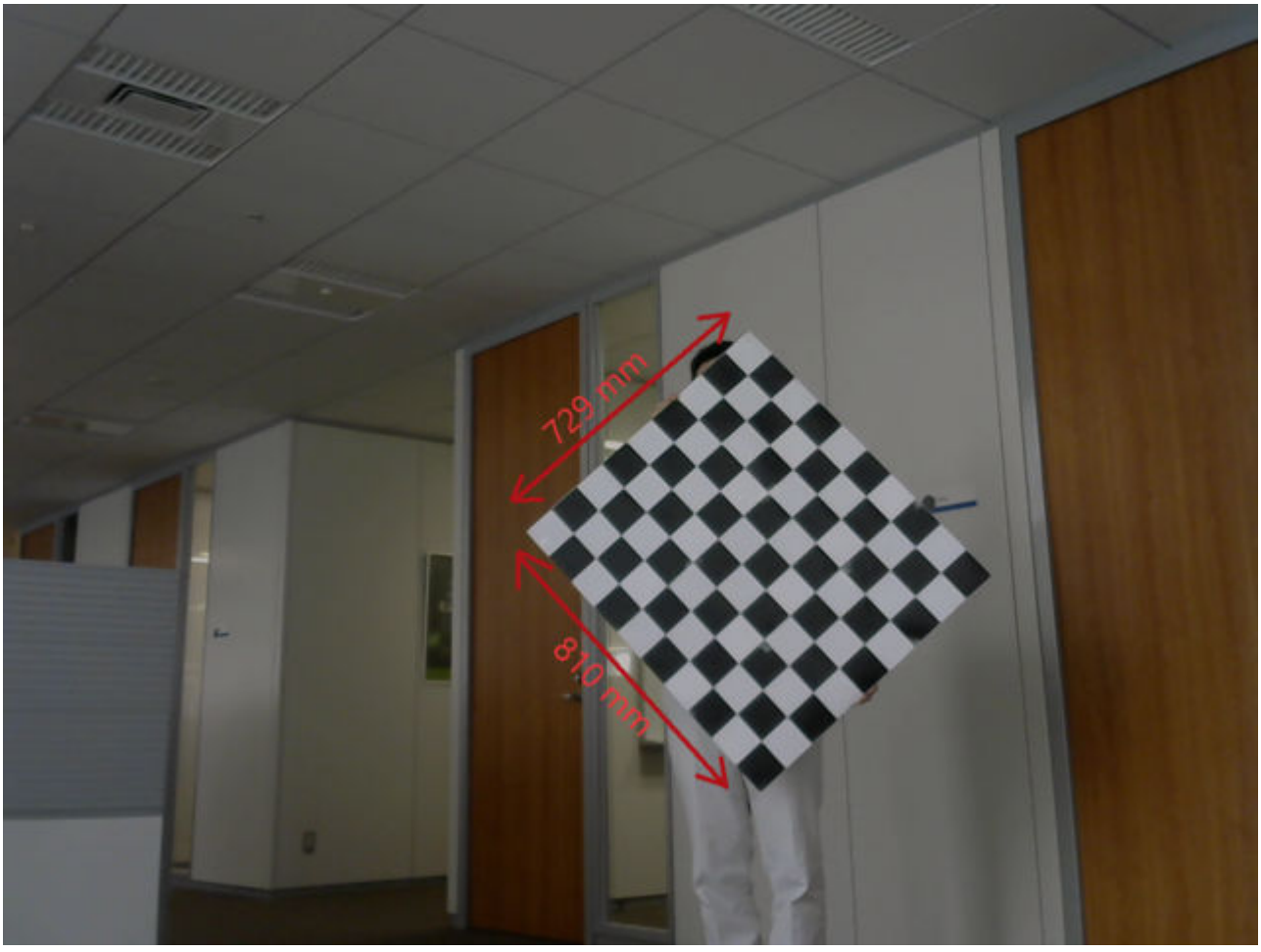
- “Calibration Guidelines”
- “Coordinate Systems in Lidar Toolbox”
- “Get Started with Lidar Camera Calibrator”

Calibration Guidelines

These guidelines help you achieve accurate results for lidar-camera calibration. For more information on lidar-camera calibration, see “What Is Lidar-Camera Calibration?”

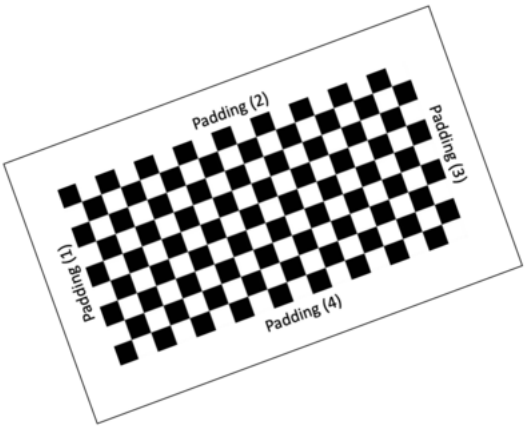
Checkerboard Guidelines

- When using the `checkerboard` function to create a checkerboard image:
  - Create a rectangular checkerboard that contains an even number of squares along one edge and an odd number of squares along the adjacent edge. This ensures a pattern with two black corners and two white corners. Patterns in which all corners are the same color can produce unexpected results for camera extrinsic parameters.
  - You can use the length differences of sides and the different corner colors to determine the orientation and the origin of the checkerboard. The **Lidar Camera Calibrator** app assigns the *x*-direction to the longer side of the checkerboard.
- Print the checkerboard from end-to-end on a foam board, as shown in this figure, to avoid any measurement errors.



- Accurately measure any padding you add along each side of the checkerboard. Padding values must be specified as a vector to the **Lidar Camera Calibrator** app or

`estimateCheckerboardCorners3d` function when you estimate the checkerboard corners. This figure shows the order of elements of the padding vector, clockwise from the left side of the checkerboard.



Checkerboard Padding

Guidelines for Capturing Data

- Capture data from both sensors simultaneously with no motion blur effects. Motion blur can degrade the accuracy of the calibration. If you are working with a video recording, carefully capture the point clouds corresponding to the respective image frames.
- The checkerboard should point towards the front axes of the camera (*z*-axis) and lidar sensor (*x*-axis).
- Hold the checkerboard target with your arms fully extended, rather than close to your body. Otherwise, parts of your body may appear to be planar with the target. This can cause inaccurate checkerboard detection.
- For sparse lidar sensors, hold the target from behind, rather than on the edges, because the `detectRectangularPlanePoints` function searches for the checkerboard plane in each cluster of the input point cloud. To further reduce false detections, specify the approximate checkerboard position using the `"ROI"` name-value argument.
- Be aware of the viewing angle or the field of view of the lidar sensor. Do not place the board in the blindspots of the sensor.
- Pay close attention to the distance between the sensor and the checkerboard. Low resolution lidar sensors, such as the Velodyne VLP-16, can have trouble accurately detecting distant checkerboards.
- Remove other items from the checkerboard plane to avoid clustering them with the checkerboard data.
- For high-resolution lidar sensors like the HDL-64 and Ouster OS1-64, you can hold the checkerboard horizontally or vertically while capturing data. However, for best results, tilt the checkerboard to a 45-degree angle while capturing data.
- This figure shows different ways to hold the checkerboard while capturing data. Capture at least 10 frames for accurate calibration.

- You must save point cloud data in the PCD or PLY format.

- Image files can be in any standard image format supported by MATLAB.

For more details on the calibration workflow, see the “Lidar and Camera Calibration” example.

See Also

Lidar Camera Calibrator | `estimateLidarCameraTransform` | `estimateCheckerboardCorners3d` | `detectRectangularPlanePoints` | `projectLidarPointsOnImage` | `fuseCameraToLidar`

Related Examples

- “Lidar and Camera Calibration”
- “Detect Vehicles in Lidar Using Image Labels”

More About

- “What Is Lidar-Camera Calibration?”
- “Get Started with Lidar Camera Calibrator”
- “Coordinate Systems in Lidar Toolbox”

What are Organized and Unorganized Point Clouds?

Introduction

There are two types of point clouds: organized and unorganized. These describe point cloud data stored in a structured manner or in an arbitrary fashion, respectively. An organized point cloud resembles a 2-D matrix, with its data divided into rows and columns. The data is divided according to the spatial relationships between the points. As a result, the memory layout of an organized point cloud relates to the spatial layout represented by the *xyz*-coordinates of its points. In contrast, unorganized point clouds consist of a single stream of 3-D coordinates, each coordinate representing a single point. You can also differentiate these point clouds based on the shape of their data. Organized point clouds are *M*-by-*N*-by-3 arrays, with the three channels representing the *x*-, *y*-, and *z*-coordinates of the points. Unorganized point clouds are *M*-by-3 matrices, where *M* is the total number of points in the point cloud.

Unorganized to Organized Conversion

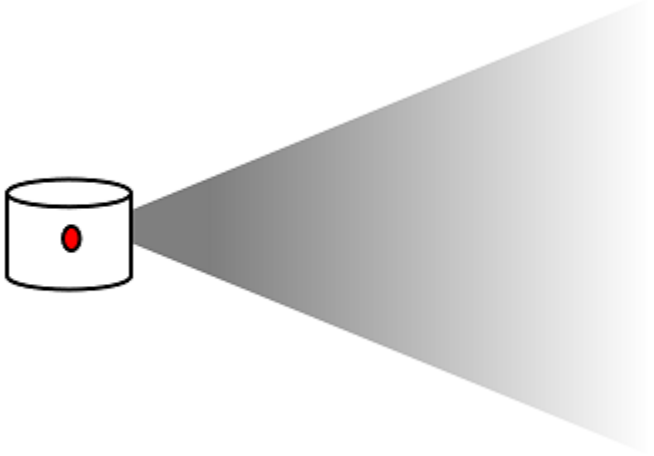
Most deep learning segmentation networks, such as SqueezeSegv1/v2, RangeNet++, and SalsaNext, process only organized point clouds. In addition, organized point clouds are used in ground plane extraction and key point detection methods. This makes organized point cloud conversion an important preprocessing step for many Lidar Toolbox workflows.

You can convert unorganized point clouds to organized point clouds by using the `pcorganize` function. The underlying algorithm uses spherical projection to represent the 3-D point cloud data in a 2-D (organized) form. It requires certain corresponding lidar sensor parameters, specified using the `lidarParameters` object, in order to convert the data.

Lidar Sensor Parameters

The sensor parameters required for conversion differ based on whether the lidar sensor has a uniform beam or a gradient beam configuration. A lidar sensor is created by stacking laser scanners vertically. Each laser scanner releases a laser pulse and rotates to capture a 3-D point cloud.

When the laser scanners are stacked with equal spacing, the lidar sensor has a uniform beam (laser scanner) configuration.

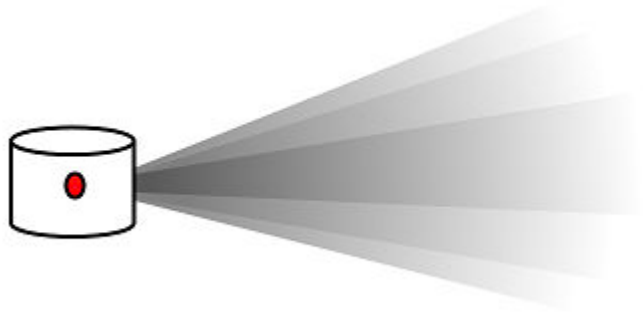


To convert unorganized point clouds captured using a lidar sensor with a uniform beam configuration, you must specify these parameters from the sensor handbook:

- Vertical resolution — Number of channels in the vertical direction, consisting of the number of lasers. Typical values include 32 and 64.
- Horizontal resolution — Number of channels in the horizontal direction. Typical values include 512 and 1024.
- Vertical field of view — Vertical field of view, in degrees. The sensor in the preceding picture has a vertical field of view of 45 degrees.

For an example, see “Create a Lidar Parameters Object”.

When the beams at the horizon are tightly packed, and those toward the top and bottom of the sensor field of view are more spaced out, the lidar sensor has a gradient beam configuration.



To convert unorganized point clouds captured using a lidar sensor with a gradient beam configuration, you must specify these parameters from the sensor handbook:

- Horizontal resolution — Number of channels in the horizontal direction. Typical values include 512 and 1024.
- Vertical beam angles — Angular position of each vertical channel, in degrees.

For an example, see “Create Lidar Parameters Object for Gradient Lidar Sensor”.

Supported Sensors

The `lidarParameters` object can automatically load the sensor parameters for some popular lidar sensors. These sensors are supported:

Sensor Name	Input
Velodyne HDL-64E	`'HDL64E'`
Velodyne HDL-32E	`'HDL32E'`
Velodyne VLP16	`'VLP16'`
Velodyne VLP32C	`'VLP32C'`
Velodyne VLP128	`'VLS128'`
Velodyne Puck LITE	`'PuckLITE'`
Velodyne Puck Hi-Res	`'PuckHiRes'`

Sensor Name	Input
Ouster OS0-32	`OS0-32`
Ouster OS0-64	`OS0-64`
Ouster OS0-128	`OS0-128`
Ouster OS1Gen1-32	`OS1Gen1-32`
Ouster OS1Gen1-64	`OS1Gen1-64`
Ouster OS1Gen1-128	`OS1Gen1-128`
Ouster OS1Gen2-32	`OS1Gen2-32`
Ouster OS1Gen2-64	`OS1Gen2-64`
Ouster OS1Gen2-128	`OS1Gen2-128`
Ouster OS2-32	`OS2-32`
Ouster OS2-64	`OS2-64`
Ouster OS2-128	`OS2-128`

See Also

`pcorganize` | `lidarParameters`

Related Examples

- “Unorganized to Organized Conversion of Point Clouds Using Spherical Projection”

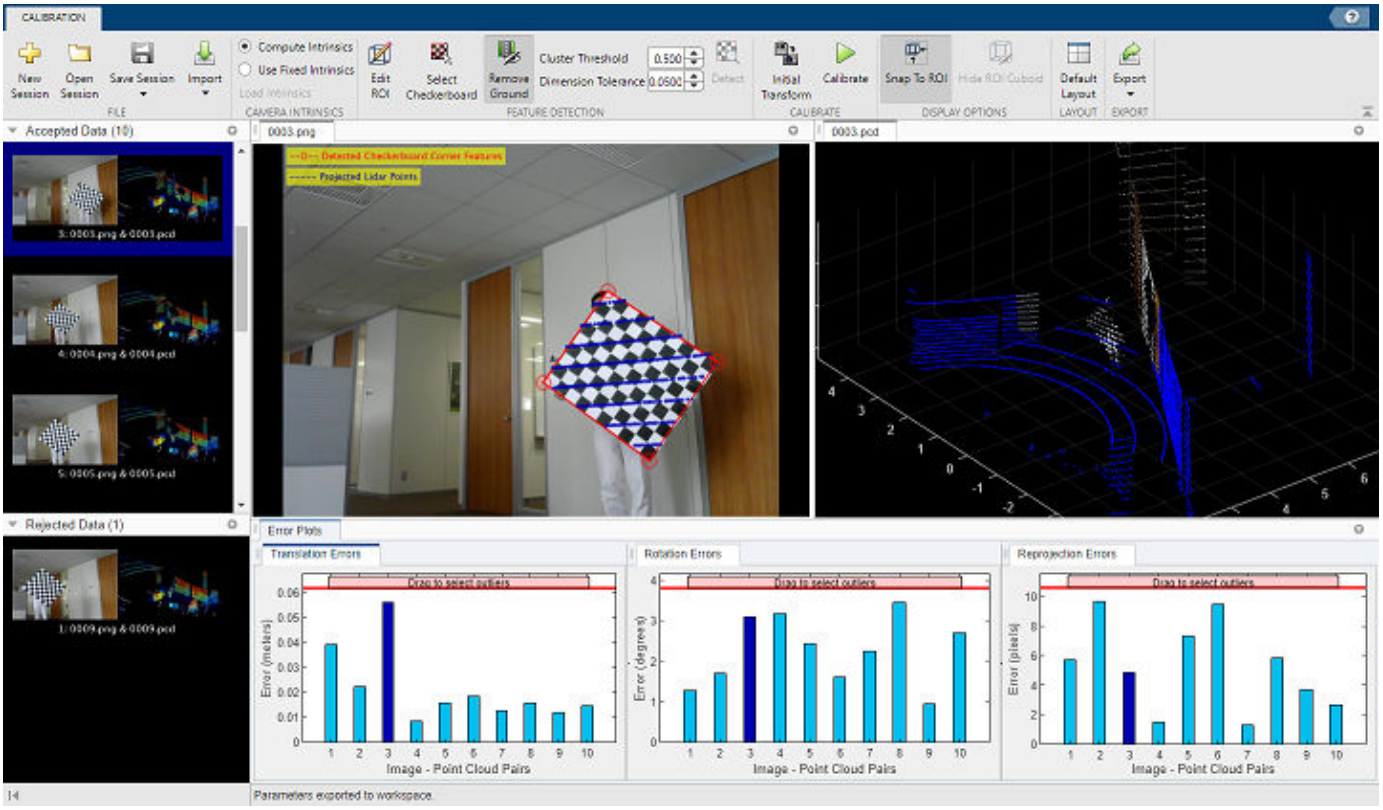
Parameter Tuning for Ground Segmentation

The `segmentGroundSMRF` function segments ground points in a point cloud. The function parameters need to be tuned in order to get accurate results based on the data. This page will explain the significance of the function parameters and how it can be tuned for aerial and driving scenario point clouds. By default, the function is tuned for aerial point cloud data.

The meaning of each parameter and the underlying are explained in the `segmentGroundSMRF` documentation. The effect of each parameter on the data is explained in the following list:

Get Started with Lidar Camera Calibrator

The **Lidar Camera Calibrator** app enables you to interactively estimate the rigid transformation between a lidar sensor and a camera.



This topic shows you the **Lidar Camera Calibrator** app workflow, as well as features you can use, to analyze and improve your results. The first, and most important, part of the calibration process is to obtain accurate and useful data. For guidelines and tips for capturing data, see “Calibration Guidelines”.

Load Data

To open the **Lidar Camera Calibrator** app, at the MATLAB command prompt, enter this command.

```
lidarCameraCalibrator
```

Alternatively, you can open the app from the **Apps** tab, under **Image Processing and Computer Vision**.

The app opens to an empty session. The app reads point cloud data in the PLY and point cloud data (PCD) formats, and images in any format supported by `imformats`. If your data is stored in a rosbag file, see the “Read Lidar and Camera Data from Rosbag File” tutorial to convert it accordingly.

Load the calibration data into the app.

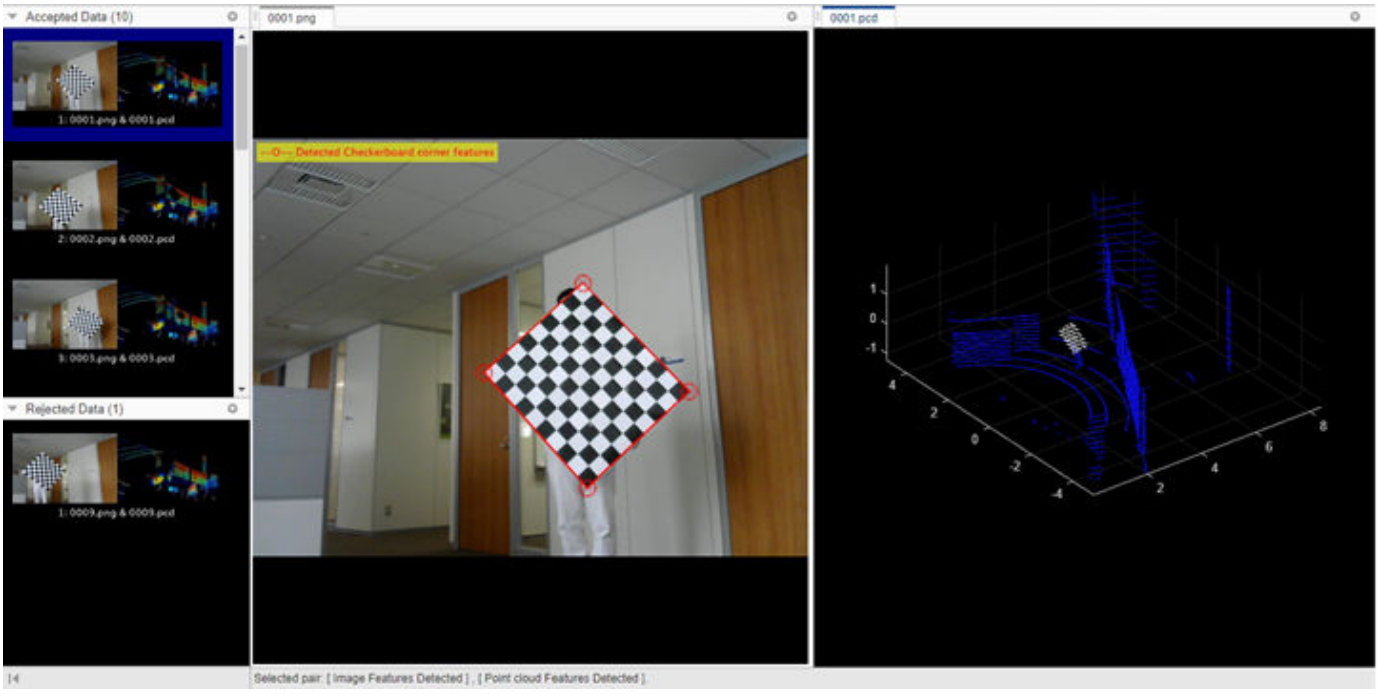
1 On the app toolstrip, select **Import** > **Import Data**, opening the **Import Data** dialog box.

2. In the **Folder for images** box, enter the path to the folder that contains the image files you want to load. Alternatively, select the **Browse** button next to the box, navigate to the folder containing the images,and click **Select Folder**.
3. In the **Folder for point clouds** box, enter the path to the folder that contains the sequence of PCD or PLY files you want to load. Alternatively, select the **Browse** button next to the box, navigate to the folder containing the files,and click **Select Folder**.
4. In the **Checkerboard Settings** section, enter the calibration checkerboard parameters. Specify the size for each checkerboard square in the **Square Size** box, and select the units of measurement from the list next to the box.
5. In the **Padding** box, enter the padding values for the checkerboard. For more information on padding, see “Checkerboard Padding”. Click **OK** to import your data.

To add more images and point clouds to the session at any point in the session, select **Import > Add Data to Session**.

Feature Detection

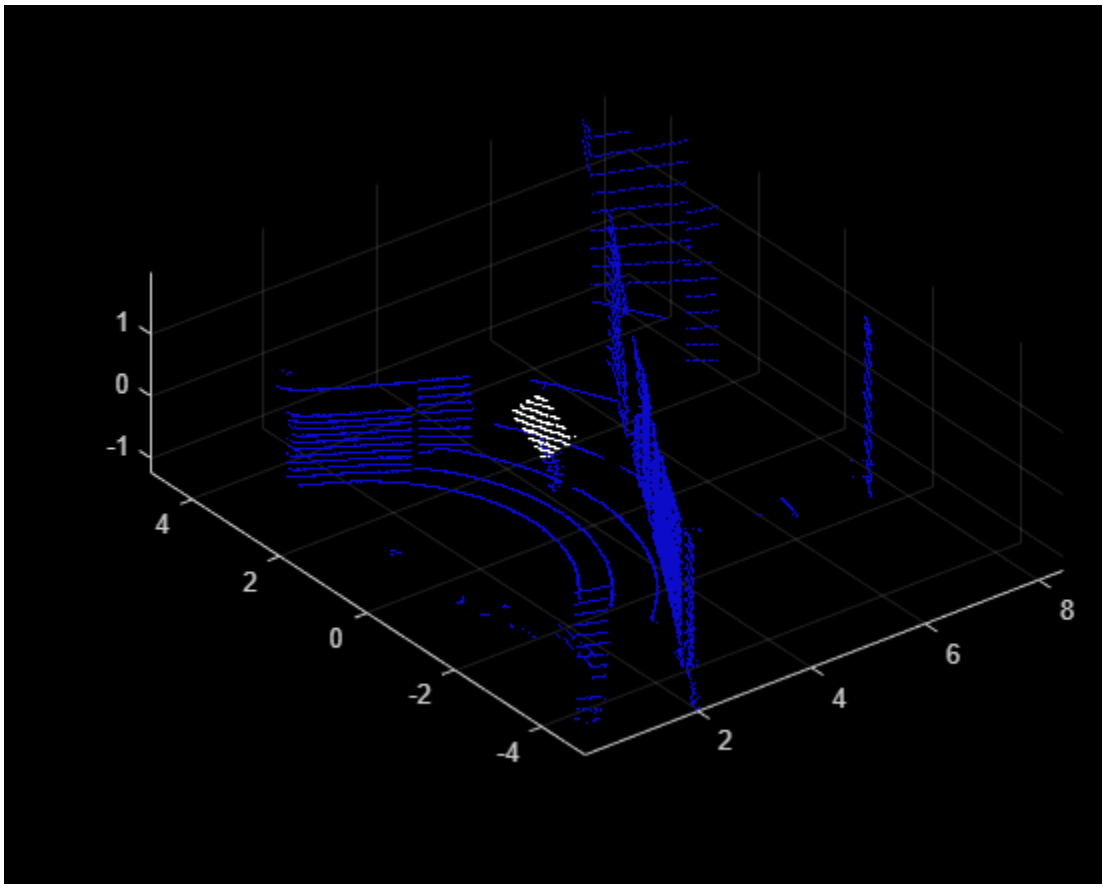
The app loads the image and point cloud data and performs an automatic feature detection pass on the data using the specified checkerboard parameters. It detects the checkerboard corners from the image data and the checkerboard plane from the point cloud data. The app interface displays the detection progression and results.



The **Accepted Data** pane displays the image and point cloud pairs that the app accepts for calibration. The app accepts an image or point cloud if it detects checkerboard features in both of them. The app uses file names to pair the image and point cloud data. It compares images to the corresponding point clouds with the same file name. The **Rejected Data** pane displays the data pairs for which the app could not detect features in the image, the point cloud, or both. You can use “Select Region of Interest” or “Select Checkerboard Region” on the **Rejected Data** to obtain better detections.

The app displays the data in the visualization area as separate panes for image and point cloud data. Each pane has tabs with the image or point cloud data file name. You can select a data pair from the **Accepted Data** or **Rejected Data** pane to visualize it in this area. When you select a data pair, the app highlights it in blue. To delete the selected data pair, press **Backspace** (PC) or **delete** (Mac). For more keyboard shortcuts for the data browser, see “Data Browser”.

The image display pane shows the image from the selected pair and the detected checkerboard corners. To detect the checkerboard corners, the app uses the `estimateCheckerboardCorners3d` function. The app computes camera intrinsics to perform feature detection. If you have camera intrinsic values, you can load them into the app, in the **Camera Intrinsics** section, by selecting **Use Fixed Intrinsics**. In the dialog box that opens, browse to your camera intrinsics file and load it into the app. After loading, in the **Feature Detection** section, select **Detect** to detect features with the new intrinsics.

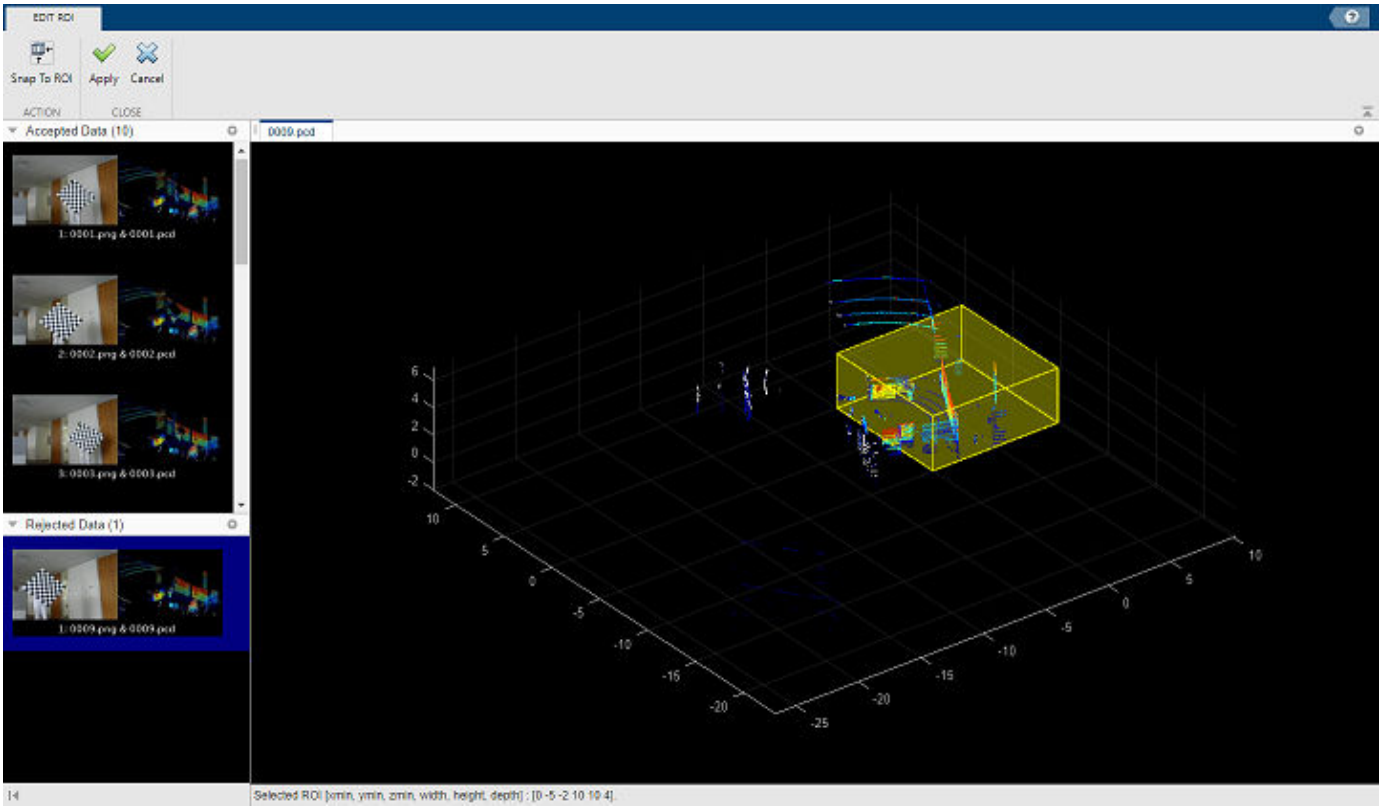


The point cloud display pane shows the point cloud from the selected pair, with the detected checkerboard plane rendered in white. To detect the plane, the app uses the `detectRectangularPlanePoints` function.

Use the various display options for point clouds in the app to improve visualization and detection.

Select Region of Interest

Select **Snap To ROI** to visualize a particular region of interest (ROI) in the point cloud. The app sets a default value for the ROI, but you can set a custom ROI using the **Edit ROI** tool. This tool enables you to manually pinpoint the region of the point cloud. Specifying an ROI to the region where the checkerboard is present can reduce data rejections and improve performance by focusing feature detection on a specific region.

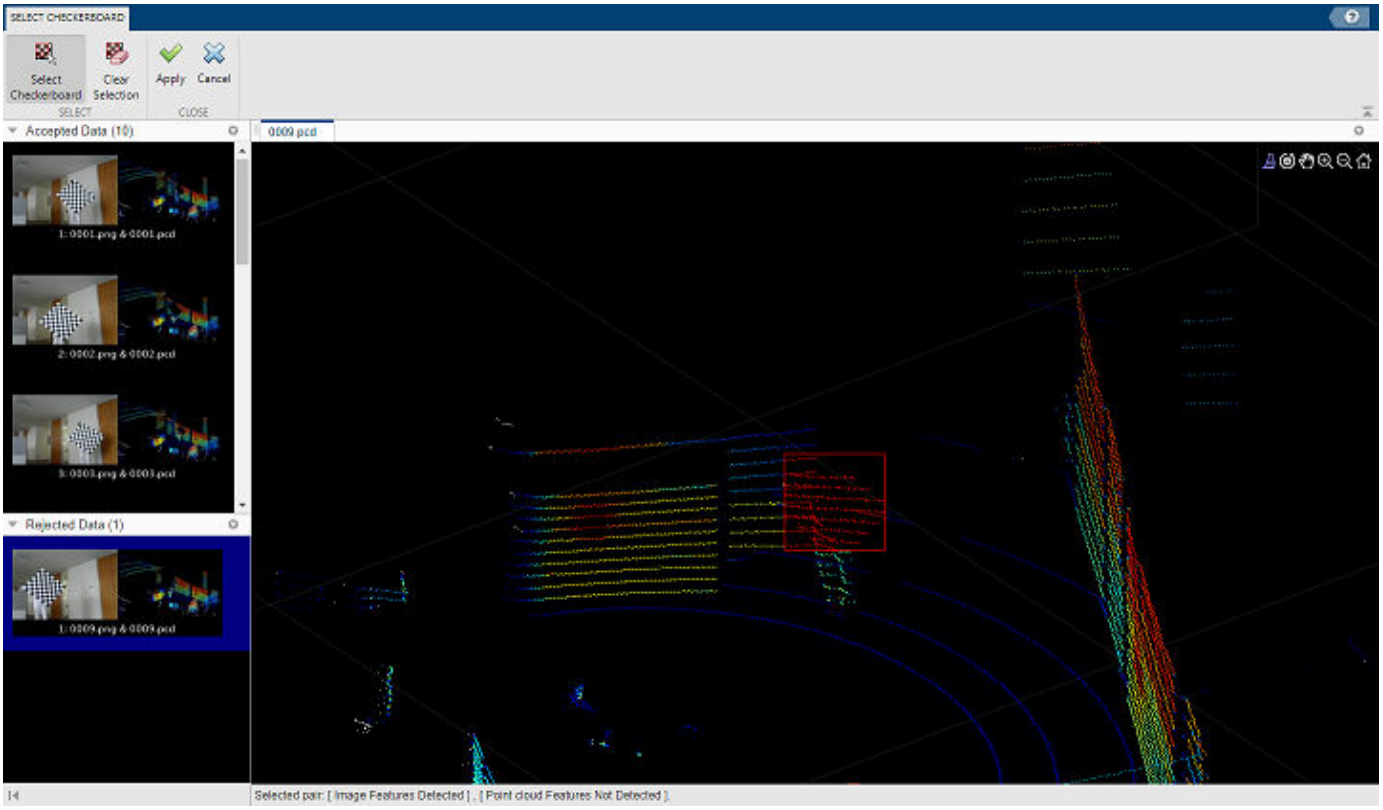


1. Select **Edit ROI**, which opens the **Edit ROI** tab. The tab contains the same **Accepted Data** and **Rejected Data** panes as the **Calibration** tab, but the point cloud display pane takes up the rest of the window.
2. Select a data pair, which the app highlights in blue. You can select a rejected data pair to tune the ROI.
3. Clear the **Snap To ROI** button to view the whole point cloud.
4. The ROI is highlighted in yellow. Point to the ROI, and the cursor turns into a hand symbol.
5. Click on any side of the ROI and drag it to toggle the size. You can select **Snap To ROI** to view the contents of the ROI and change the size accordingly.
6. Select **Apply** to save your changes or **Cancel** to discard them.

Because the app applies the new ROI on all the point cloud frames, you must define an ROI that covers all areas in which you placed the checkerboard. Clear the **Snap To ROI** button to view the whole point cloud and select the **Hide ROI Cuboid** to remove the ROI highlighting. Select **Detect** to detect features in the selected ROI. Alternatively, you can use keyboard shortcuts to perform these tasks. For more information, see “Edit ROI”.

Select Checkerboard Region

To further tune the detections, you can use the **Select Checkerboard** feature to manually select checkerboard points in any point cloud frame.

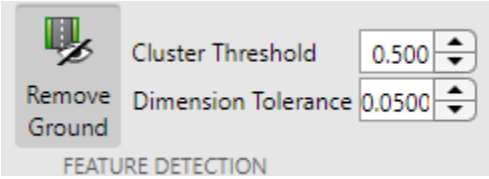


1. On the app toolstrip, select **Select Checkerboard**. The app opens the **Select Checkerboard** tab. This tab contains the same **Accepted Data** and **Rejected Data** panes as the **Calibration** tab, but the point cloud display pane takes up the rest of the window.
2. Select a data pair. The app highlights it in blue. You can select a rejected data pair and find the checkerboard in the point cloud.
3. Use the zoom and rotate options in the axes toolbar of the point cloud display to locate the checkerboard.
4. Select **Select Checkerboard**. The cursor changes into a crosshair.
5. Click and drag the cursor over the checkerboard. A selection rectangle appears, with the points inside it highlighted in red. Alternatively, you can also select **Brush/Select Data** on the axes toolbar.
6. After selecting the points, rotate the point cloud to check whether any background points have been selected. If your selection contains unwanted points, select **Clear Selection** to start over.
7. Select **Apply** to save the selected points, or **Cancel** to discard them.

This checkerboard selection applies only to the current point cloud. Select **Detect** to detect features using the manually selected checkerboard.

Feature Detection Settings

The app provides these feature detection settings in which you can tune parameters.



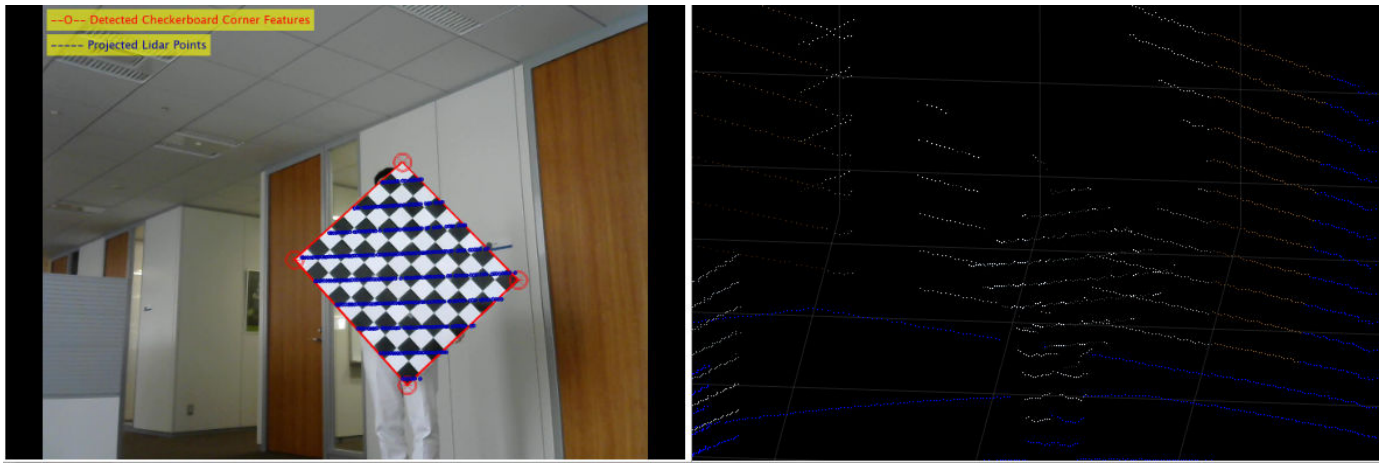
- **Remove Ground** — Remove ground points from the point cloud. The app uses the `pcfitplane` function to estimate the ground plane. The **Remove Ground** feature is enabled by default. Select **Remove Ground** to clear it.
- **Cluster Threshold** — Clustering threshold for two adjacent points in the point cloud, specified in meters. The clustering process is based on the Euclidean distance between adjacent points. If the distance between two adjacent points is less than the clustering threshold, both points belong to the same cluster. Low-resolution lidar sensors require a higher **Cluster Threshold**, while high-resolution lidar sensors benefit from a lower **Cluster Threshold**.
- **Dimension Tolerance** — Tolerance for uncertainty in the rectangular plane dimensions, specified in the range [0,1]. A higher **Dimension Tolerance** indicates a more tolerant range for the rectangular plane dimensions.

Select **Detect** to detect features using the new parameters.

Calibration

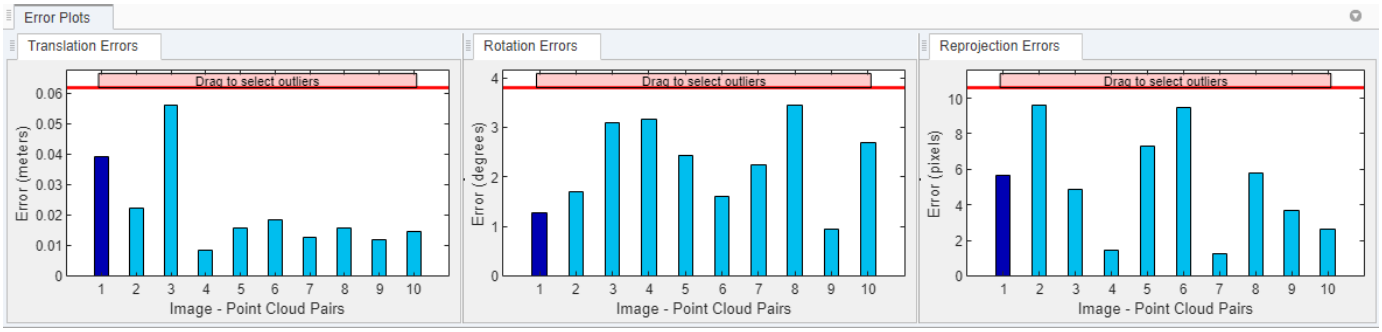
When you are satisfied with the detection results, select **Calibrate** button to calibrate the sensors. If you have an estimated transformation matrix, select **Initial Transform** to load the transformation matrix from a file or the workspace. The app assumes the rotation angle between the lidar sensor and the camera is in the range [–45 45], in degrees, along each axis. For a rotation angle outside this range, use **Initial Transform** to specify an initial transformation to improve calibration accuracy.

After calibration, the app interface displays the image with the checkerboard points from the point cloud projected onto it. The app uses the `projectLidarPointsOnImage` function to project the lidar points onto the image. The color information of the images is fused with the point cloud data using the `fuseCameraToLidar` function.



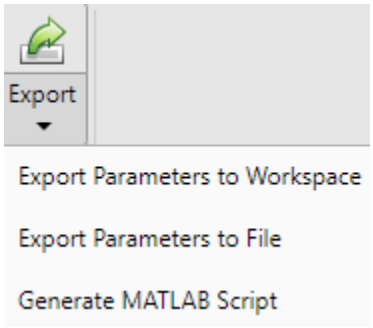
The app also provides the inaccuracy metrics for the transformation matrix using error plots. The plots specify these errors in each data pair:

- **Translation Errors** — The difference between the centroid coordinates of the checkerboard planes in the point clouds and those in the corresponding images. The app returns the error values in meters.
- **Rotation Errors** — The difference between the normal angles defined by the checkerboard planes in the point clouds and those in the corresponding images. The app estimates the plane in the image using the checkerboard corner coordinates. The app returns the error values in degrees.
- **Reprojection Error**— The difference between the projected (transformed) centroid coordinates of the checkerboard planes from the point clouds and those in the corresponding images. The app returns the error values in pixels.



When you select a data pair in the data browser, the corresponding bars in the error plot are highlighted in dark blue. You can tune the calibration results by removing outliers. Drag the red line on each plot vertically to set error limits. The app selects all the data pairs with an error value greater than error limit as outliers, and highlights the error bars and their corresponding data pairs in the data browser in blue. Right-click any of the selected data pairs on the data browser and select **Remove and Recalibrate** to delete the outliers and recalibrate the sensors. Deleting outliers can improve calibration accuracy. For a list of keyboard shortcuts to use with the error plots, see “Error Plots”.

Export Results



You can export the transformation matrix and error metrics, as variables, into the workspace or a MAT-file. You can generate a MATLAB script of the complete app workflow to use in your projects.

Keyboard Shortcuts and Mouse Actions

Note On Macintosh platforms, use the **Command (⌘)** key instead of **Ctrl**.

Use keyboard shortcuts and mouse actions to increase productivity while using the **Lidar Camera Calibrator** app.

Data Browser

Task	Action
Navigate through data pairs in the **Accepted Data** or **Rejected Data** pane	Up or down arrow
Select all data pairs in the data browser	**Ctrl+A**
Select multiple data pairs above or below the currently selected data pair.	Hold **Shift** and press the up arrow or down arrow
Select multiple data pairs. **Note** Deselecting a selected data pair is not currently supported using the same shortcut.	Hold **Ctrl** and click on data pairs
Delete the selected data pair from the data browser.	• PC: **Backspace** or **Delete** • Mac: **delete** A dialog box appears to deletion.
Select the data pair*N* above the currently selected data pair. *N* is the number of data pairs fully displayed in the data browser at the current time.	• PC: **Page Up** • Mac: Hold **Fn** and press the up arrow
Select the data pair*N* below the currently selected data pair. *N* is the number of data pairs fully displayed in the data browser at the current time.	• PC: **Page Down** • Mac: Hold **Fn** and press the down arrow
Select the first data pair in the data browser.	• PC: **Home** • Mac: Hold **Fn** and press the left arrow
Select the last data pair in the data browser.	• PC: **End** • Mac: Hold **Fn** and press the right arrow

Error Plots

Use these shortcuts on the error plots to analyze the data. Operations on any of the three error plots affect the corresponding error bars on all three plots.

Task	Action
Select the error bar left of the currently selected error bar.	Left arrow
Select the error bar right of the currently selected error bar.	Right arrow
Select the error bar right or left of the currently selected error bar, in addition to the currently selected error bar.	Hold **Shift** and press the left arrow or right arrow

Task	Action
Select multiple error bars. **Note** Deselecting a selected error bar is not currently supported using the same shortcut.	Hold **Ctrl** and click error bars
Select all error bars.	**Ctrl+A**
Delete the selected error bar and corresponding data pair from the data browser. The app then recalibrates the sensors.	• PC: **Backspace** or **Delete** • Mac: **delete** A dialog box appears to confirm deletion.

Edit ROI

Shortcuts to use on the **Edit ROI** tab.

Task	Action
Undo ROI size change. **Note** The app stores only the last three sizes of the ROI, so you cannot undo more than three times in a row.	**Ctrl+Z**
Redo ROI size change.	**Ctrl+Y**
Clear ROI selection	**Esc**

Limitations

The **Lidar Camera Calibrator** app has these limitations:

- The script generated from **Export > Generate MATLAB Script** does not include any checkerboard regions manually selected using the **Select Checkerboard** feature. In the script, the checkerboard region is detected in the specified ROI.
- After manually selecting checkerboard regions using the **Select Checkerboard** feature, when you return to the **Calibration** tab, you can see the selected points (highlighted in red) only while viewing the whole point cloud (when **SnapToROI** is cleared).

See Also

Lidar Camera Calibrator | `estimateCheckerboardCorners3d` | `estimateLidarCameraTransform` | `projectLidarPointsOnImage` | `fuseCameraToLidar` | `bboxCameraToLidar`

Related Examples

- “Lidar and Camera Calibration”
- “Read Lidar and Camera Data from Rosbag File”
- “Detect Vehicles in Lidar Using Image Labels”

More About

- “What Is Lidar-Camera Calibration?”
- “Calibration Guidelines”

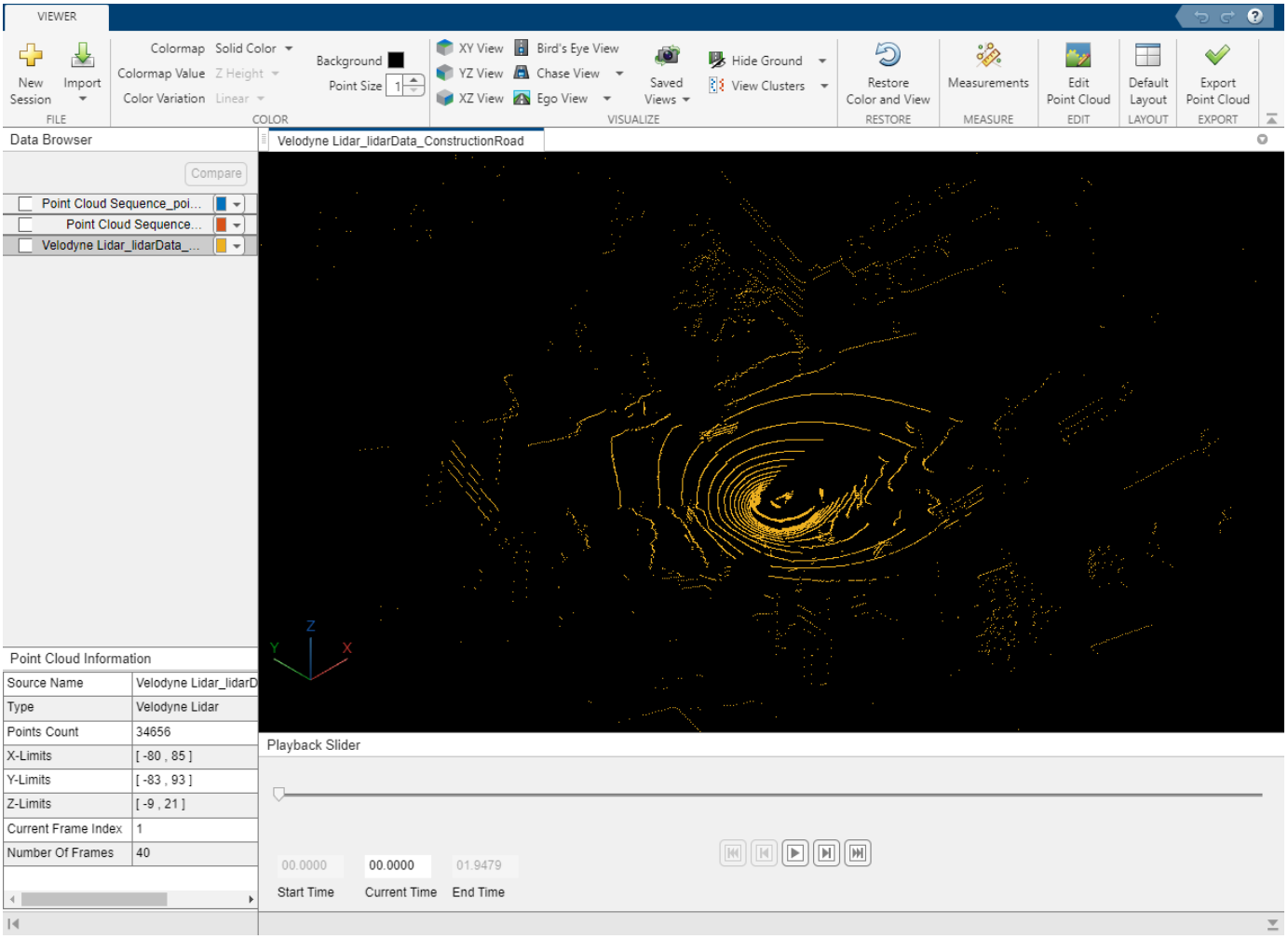
Get Started with Lidar Viewer

The **Lidar Viewer** app is a tool to visualize, analyze, and process point cloud data. You can also use this app to preprocess your data for workflows such as labeling, segmentation, and calibration.

To open the app, enter this command in the MATLAB command window.

```
lidarViewer
```

Alternatively, you can open the app from the **Image Processing and Computer Vision** section in the **Apps** tab.



Use these steps to load, visualize, analyze, and edit point clouds using the app.

1. “Import, View, and Export Point Cloud”
2. “Visualize Point Cloud Using Color and Camera Views”
3. “Measure Point Cloud”
4. “Edit Point Cloud”

See Also

Apps
Lidar Viewer | **Lidar Labeler**

Functions
`pcshow` | `pointCloud` | `pcdownsample` | `pcmedian` | `pcdenoise` | `pcorganize` | `segmentGroundSMRF` | `pcfitplane` | `segmentGroundFromLidarData`

Objects
`pointCloud` | `lasFileReader`

More About

- “Introduction to Lidar Processing”
- “What are Organized and Unorganized Point Clouds?”

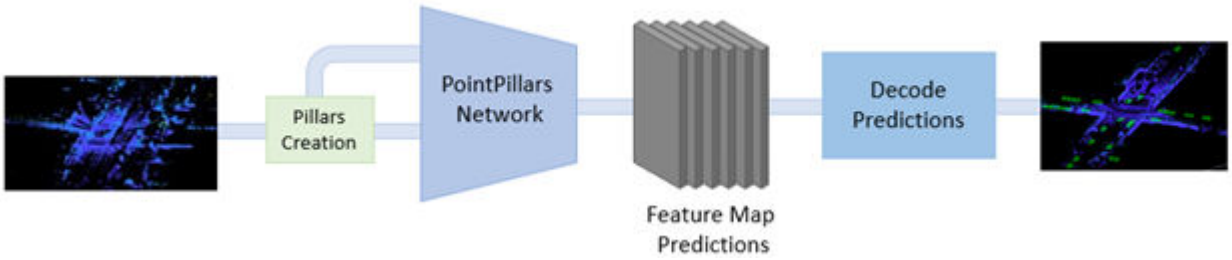
See Also

Related Examples

- “Create Custom Preprocessing Workflow with Lidar Viewer”
- “Transform Point Cloud Using Lidar Viewer”

Getting Started with PointPillars

PointPillars is a method for 3-D object detection using 2-D convolutional layers. PointPillars network has a learnable encoder that uses PointNets to learn a representation of point clouds organized in pillars (vertical columns). The network then runs a 2-D convolutional neural network (CNN) to produce network predictions, decodes the predictions, and generates 3-D bounding boxes for different object classes such as cars, trucks, and pedestrians.



The PointPillars network has these main stages.

1. Use a feature encoder to convert a point cloud to a sparse pseudoimage.
2. Process the pseudoimage into a high-level representation using a 2-D convolution backbone.
3. Detect and regress 3D bounding boxes using detection heads.

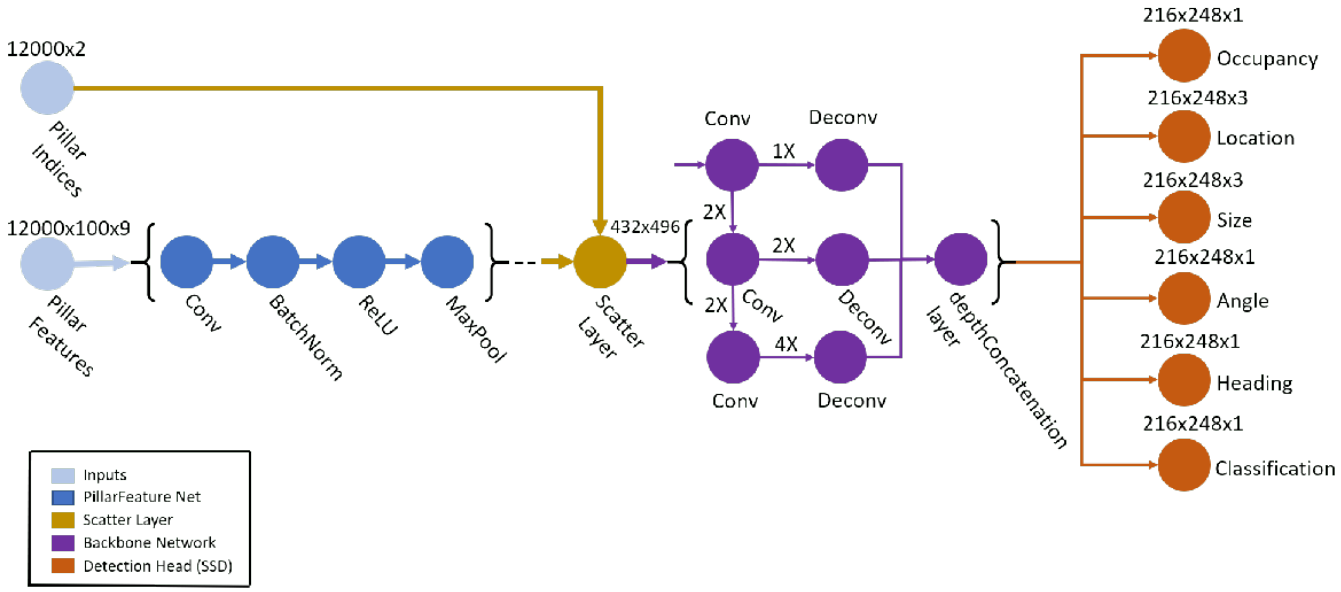
PointPillars Network

A PointPillars network requires two inputs: pillar indices as a *P*-by-2 and pillar features as a *P*-by-*N*-by-*K* matrix. *P* is the number of pillars in the network, *N* is the number of points per pillar, and *K* is the feature dimension.

The network begins with a feature encoder, which is a simplified PointNet. It contains a series of convolution, batch-norm, and relu layers followed by a max pooling layer. A scatter layer at the end maps the extracted features into a 2-D space using the pillar indices.

Next, the network has a 2-D CNN backbone that consists of encoder-decoder blocks. Each encoder block consists of convolution, batch-norm, and relu layers to extract features at different spatial resolutions. Each decoder block consists of transpose convolution, batch-norm, and relu layers.

The network then concatenates output features at the end of each decoder block, and passes these features through six detection heads with convolutional and sigmoid layers to predict occupancy, location, size, angle, heading, and class.



Create PointPillars Network

You can use the **Deep Network Designer** app to interactively create a PointPillars deep learning network. To programmatically create a PointPillars network, use the `pointPillarsObjectDetector` object.

Transfer Learning

Reconfigure a pretrained PointPillars network by using the `pointPillarsObjectDetector` object to perform transfer learning. Specify the new object classes and the corresponding anchor boxes to train the network on a new dataset.

Train PointPillars Object Detector and Perform Object Detection

Use the `trainPointPillarsObjectDetector` function to train a PointPillars network. To perform object detection on a trained PointPillars network, use the `detect` function. For more information on how to train a PointPillars network, see “Lidar 3-D Object Detection Using PointPillars Deep Learning”.

Code Generation

To learn how to generate CUDA® code for a PointPillars Network, see “Code Generation For Lidar Object Detection Using PointPillars Deep Learning”.

References

[1] Lang, Alex H., Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. “PointPillars: Fast Encoders for Object Detection From Point Cloud” In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 12689–97. Long Beach, CA, USA: IEEE, 2019. https://doi.org/10.1109/CVPR.2019.01298.

[2] Hesai and Scale. PandaSet. https://scale.com/open-datasets/pandaset.

See Also

Apps
Deep Network Designer | Lidar Viewer | Lidar Labeler

Objects
`pointPillarsObjectDetector`

Functions
`trainPointPillarsObjectDetector` | `detect`

Related Examples

- “Lidar 3-D Object Detection Using PointPillars Deep Learning”
- “Code Generation For Lidar Object Detection Using PointPillars Deep Learning”
- “Lane Detection in 3-D Lidar Point Cloud”
- “Unorganized to Organized Conversion of Point Clouds Using Spherical Projection”

More About

- “Deep Learning in MATLAB” (Deep Learning Toolbox)
- “Getting Started with Point Clouds Using Deep Learning”

Getting Started with PointNet++

PointNet++ is a popular neural network used for semantic segmentation of unorganized lidar point clouds. Semantic segmentation associates each point in a 3-D point cloud with a class label, such as car, truck, ground, or vegetation.

PointNet++ network partitions the input points into a set of clusters and then extracts the features using a multi-layer perceptron (MLP) network. The network applies PointNet recursively on the nested, partitioned inputs to extract multi-scale features for accurate semantic segmentation.

Applications of PointNet++ include:

- Tree segmentation for digital forestry applications.
- Extracting a digital terrain model from aerial lidar data.
- Perception for indoor navigation in robotics.
- 3-D city modelling from aerial lidar data.

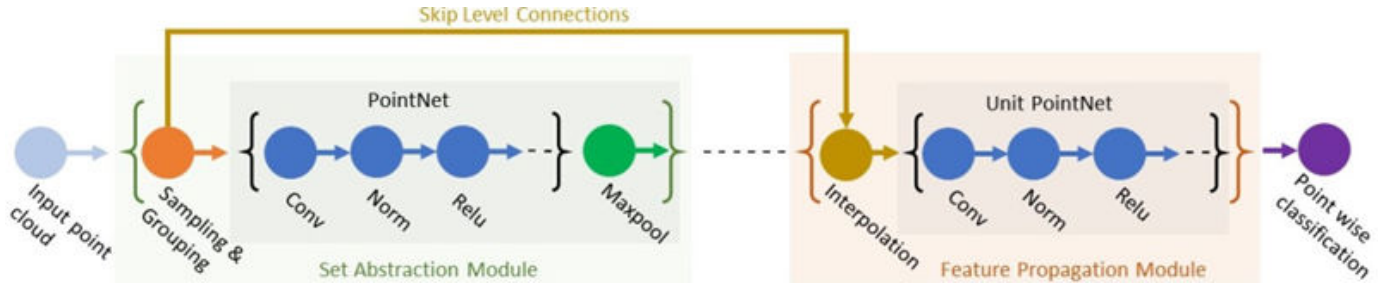
PointNet++ Network

The PointNet++ network contains an encoder with set abstraction modules and a decoder with feature propagation modules.

The set abstraction module processes and extracts a set of points to produce a new set with fewer elements. Each set abstraction module contains a sampling and grouping layer followed by a mini-PointNet network.

- The sampling and grouping layer performs sampling by identifying the centroids of local regions. It then performs grouping by constructing local region sets of the neighboring points around the centroids.
- The mini-PointNet network contains a shared MLP network with a series of convolution, normalization, relu layers followed by a max pooling layer. It encodes the local region patterns into feature vectors.

The feature propagation module interpolates the subsampled points and then concatenates them with the point features from the set abstraction modules. The network then passes these features through the unit PointNet network.



The sampling & grouping layer of the set abstraction module and the interpolation layer of the feature propagation module in this network are implemented using the `functionLayer` function.

Create PointNet++ Network

Use the `pointnetplusLayers` function to create a PointNet++ network for segmenting point cloud data.

Train PointNet++ Network

To learn how to train a PointNet++ network for segmenting point cloud data, see “Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”.

Code Generation

To learn how to generate CUDA® code for a PointNet++ network, see “Code Generation For Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”.

References

[1] Qi, Charles R., Li Yi, Hao Su, and Leonidas J. Guibas. ‘PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space’. *ArXiv:1706.02413 [Cs]*, 7 June 2017. https://arxiv.org/abs/1706.02413.

[2] Varney, Nina, Vijayan K. Asari, and Quinn Graehling. ‘DALES: A Large-Scale Aerial LiDAR Data Set for Semantic Segmentation’. *ArXiv:2004.11985 [Cs, Stat]*, 14 April 2020. https://arxiv.org/abs/2004.11985.

See Also

Apps
Deep Network Designer | Lidar Viewer | Lidar Labeler

Functions
`pointnetplusLayers` | `squeezesegv2Layers` | `semanticseg` | `trainNetwork` | `evaluateSemanticSegmentation`

Related Examples

- “Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”
- “Code Generation For Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”
- “Lidar Point Cloud Semantic Segmentation Using SqueezeSegV2 Deep Learning Network”
- “Lidar Point Cloud Semantic Segmentation Using PointSeg Deep Learning Network”

More About

- “Deep Learning in MATLAB” (Deep Learning Toolbox)
- “Getting Started with Point Clouds Using Deep Learning”

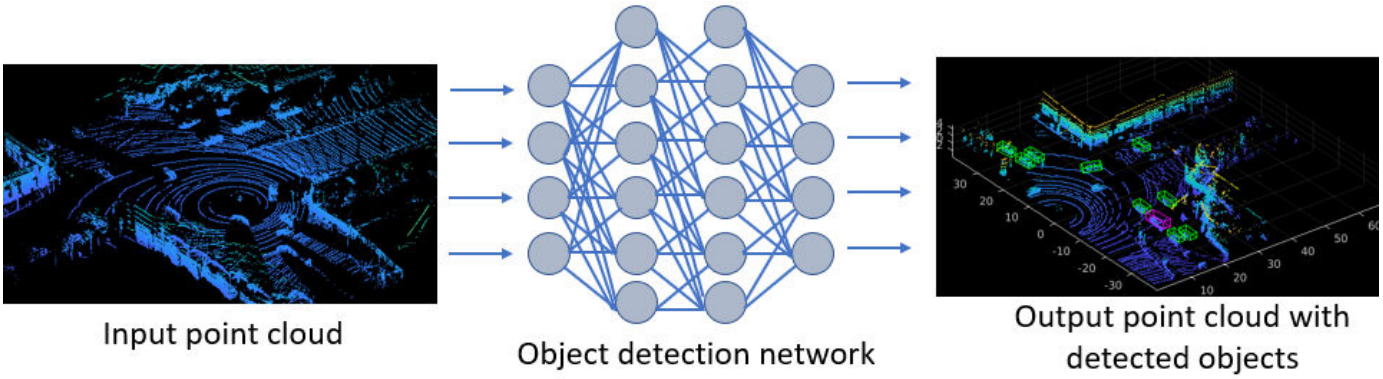
Object Detection in Point Clouds Using Deep Learning

3-D object detection has great significance in autonomous navigation, robotics, medicine, remote sensing, and augmented reality applications. Though point clouds provide rich 3-D information, object detection in point clouds is a challenging task due to the sparse and unstructured nature of data.

Using deep neural networks to detect objects in a point cloud provides fast and accurate results. A 3-D object detector network takes point cloud data as an input and produces 3-D bounding boxes around each detected object.

These are few popular methods of object detection based on the network input.

- Input different point cloud views such as Bird's-eye-view (BEV), front-view, or image view to a network and regress 3-D bounding boxes. You can also fuse features from different views for more accurate detections.
- Convert point cloud data into a more structured representation such as pillars or voxels, then apply a 3-D convolutional neural network to obtain bounding boxes. PointPillars and VoxelNet are widely popular networks using this method.
  - VoxelNet converts the point cloud data into equally spaced voxels and encodes features within each voxel into a 4-D tensor. Then, obtains the detection results by using a region proposal network.
  - PointPillars network uses PointNets to learn the features of the point cloud organized into vertical pillars. The network then encodes these features as pseudo images to predict bounding boxes by using a 2-D object detection pipeline. For more information, see “Getting Started with PointPillars”.
- Preprocess point cloud data to derive a 2-D representation, use a 2-D CNN to obtain 2-D bounding boxes. Then, project these 2-D boxes onto the point cloud data to obtain 3-D detection results.



Create Training Data for Object Detection

Training the network on large labeled datasets provides faster and more accurate detection results.

Use the **Lidar Labeler** app to interactively label point clouds and export label data for training. You can label cuboids, lines, and voxel regions inside a point clouds using the app. You can also add scene labels for point classification. For more information, see “Get Started with the Lidar Labeler”.

Augment and Preprocess Data

Using data augmentation techniques adds variety to the limited datasets. You can transform point clouds by translating, rotating, and adding new bounding boxes to the point cloud. This provides distinct point clouds for training. For more details, see “Data Augmentations for Lidar Object Detection Using Deep Learning”.

To convert unorganized point clouds into organized format, use the `pcorganize` function. For more information, see the “Unorganized to Organized Conversion of Point Clouds Using Spherical Projection” example.

When your network input is 2-D, you can use the `ImageDatastore`, `PixelLabelDatastore`, and `boxLabelDatastore` objects to divide and store the training and the test data. To store point clouds, use the `fileDatastore` object.

For aerial lidar data, use the `blockedPointCloudDatastore` and `blockedPointCloud` functions, respectively to store and process point cloud data as blocks.

For more information, see

- “Preprocess Data for Domain-Specific Deep Learning Applications” (Deep Learning Toolbox)
- “Datastores for Deep Learning” (Deep Learning Toolbox)

Create Object Detection Network

Define your network based on the network input and the layers. For a list of supported layers and how to create them, see the “List of Deep Learning Layers” (Deep Learning Toolbox). To visualize the network architecture, use the `analyzeNetwork` function.

You can also design a network layer-by-layer interactively using the **Deep Network Designer**.

Use the `pointPillarsObjectDetector` object, to create a PointPillars object detector network.

Train Object Detector Network

To specify the training options, use the `trainingOptions` function and you can train the network by using the `trainNetwork` function.

Use the `trainPointPillarsObjectDetector` function to train a PointPillars network.

Detect Objects in Point Clouds Using Deep Learning Detectors and Pretrained Models

Use the `detect` function to detect objects using a PointPillars network.

To evaluate the detection results, use the `evaluateDetectionAOS` and `bboxOverlapRatio` functions.

Lidar Toolbox provides these pretrained object detection models for PointPillars and Complex YOLOv4 networks. For more information, see

- “Lidar 3-D Object Detection Using PointPillars Deep Learning”

- “Lidar Object Detection Using Complex-YOLO v4 Network”

Code Generation

To learn how to generate CUDA® code for a segmentation workflow, see these examples.

“Code Generation for Lidar Object Detection Using SqueezeSegV2 Network”

“Code Generation For Lidar Object Detection Using PointPillars Deep Learning”

See Also

Apps
Lidar Labeler | **Deep Network Designer**

Functions
`pointPillarsObjectDetector` | `trainPointPillarsObjectDetector` | `detect`

Objects
`fileDatastore` | `boxLabelDatastore` | `pixelLabelDatastore` | `blockedPointCloudDatastore`

See Also

Related Examples

- “Lidar Object Detection Using Complex-YOLO v4 Network”
- “Code Generation for Lidar Object Detection Using SqueezeSegV2 Network”
- “Lidar 3-D Object Detection Using PointPillars Deep Learning”
- “Code Generation For Lidar Object Detection Using PointPillars Deep Learning”

More About

- “Deep Learning in MATLAB” (Deep Learning Toolbox)
- “Getting Started with Point Clouds Using Deep Learning”
- “Getting Started with PointPillars”
- “Semantic Segmentation in Point Clouds Using Deep Learning”
- “Datastores for Deep Learning” (Deep Learning Toolbox)
- “List of Deep Learning Layers” (Deep Learning Toolbox)

Semantic Segmentation in Point Clouds Using Deep Learning

Segmentation is a fundamental step in processing 3D point clouds. Point cloud semantic segmentation or classification is a process of associating each point in a point cloud with a semantic label such as `tree`, `person`, `road`, `vehicle`, `ocean`, or `building`.

Segmentation clusters points with similar characteristics into homogeneous regions. These regions correspond to specific structures or objects in a point cloud scene.

These are the major approaches for point cloud semantic segmentation.

- Classify each point or a point cluster based on individual features by using feature extraction and neighborhood selection.
- Extract point statistics and spatial information from the point cloud to classify the points using statistical and contextual modeling.

Applications for segmentation include urban planning, oceanography, forestry, autonomous driving, robotics, and navigation.

Deep Learning-Based Segmentation

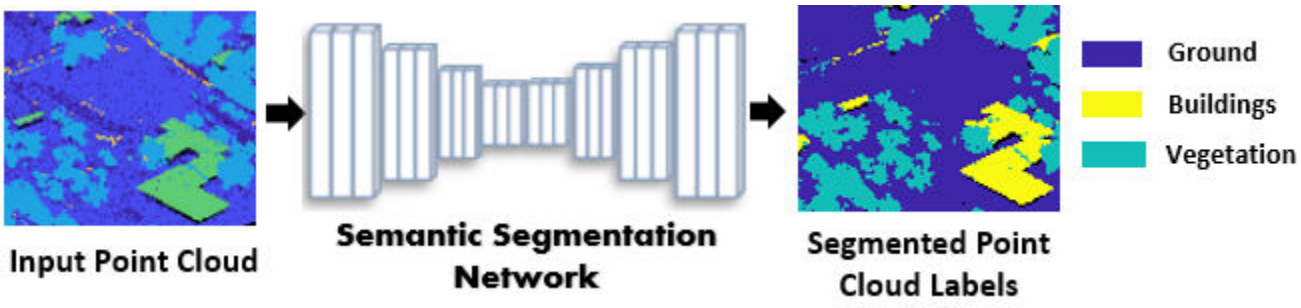
Deep learning is an efficient approach to point cloud semantic segmentation in which you train a network to segment and classify points by extracting features from the input data. You cannot apply standard convolutional neural networks (CNNs) used for image segmentation to raw point clouds due to the unordered, sparse, and unstructured nature of point cloud data. In most cases, you must transform raw point clouds before feeding them as an input to a segmentation network.

These are the categories of deep learning segmentation methods, divided by how you format input to the network.

- Multiview-based methods — Reduce 3-D point clouds to 2-D projected images and process them directly using 2-D CNNs. After classification you must postprocess the results to restore the 3-D structure.
- Voxel-based methods — Input point cloud data as voxels and use the standard 3-D CNNs. This addresses the unordered and unstructured nature of raw point cloud data.
- Point-based methods — Directly apply the deep learning network on individual points.

PointNet is the most popular point-based deep learning framework. This network uses multi-layer perceptrons (MLP) to extract local features from individual points, then aggregates all local features into global features using maxpooling. PointNet concatenates the aggregated global and local features into combined point features, and then extracts new features from the combined point features by using MLPs. The network predicts semantic labels based on the new features.

PointNet++ is improves on the basic PointNet model by additionally capturing local features hierarchically. For more information on PointNet++, see “Getting Started with PointNet++”.



Create Training Data for Semantic Segmentation

Lidar Toolbox provides functions to import and read raw point cloud data from several file formats. For more information, see “I/O”.

The toolbox enables you to divide this data into training and test data sets, and store them as datastore objects. For example, you can store point cloud files by using the `fileDatastore` object. For more information on datastore objects, see “Datastores for Deep Learning” (Deep Learning Toolbox).

The “Import Point Cloud Data For Deep Learning” example shows you how to import a large point cloud data set, and then configure and load a datastore.

Label Data

You need a large, labeled data set to train a deep learning network. If you have an unlabeled data set, you can use the **Lidar Labeler** app to label your training data. For information on how to use the app, see “Get Started with the Lidar Labeler”.

Augment Data

Data augmentation adds variety to the existing training data. The robustness of a network to data transformations increases when you train it on a data set with a lot of variety.

Augmentation techniques reduce overfitting problems and enable the network to better learn and infer features.

For more information on how to perform data augmentation on point clouds, see “Data Augmentations for Lidar Object Detection Using Deep Learning”.

Preprocess Data

You can preprocess your data before training the network. Lidar Toolbox provides function to perform various preprocessing tasks.

- Denoise, downsample, and filter point clouds.
- Convert unorganized data into the organized format.
- Divide aerial point cloud data into blocks to perform block-by-block processing.

For more information on preprocessing, see “Lidar Processing Applications” (Deep Learning Toolbox).

To interactively visualize, analyze, and preprocess point cloud data, use the **Lidar Viewer** app. For more information on how to use the app, see “Get Started with Lidar Viewer”.

Create Semantic Segmentation Network

Define your network based on the network input and the layers.

Lidar Toolbox provides these function to create segmentation networks.

- `pointnetplusLayers` — Create PointNet++ segmentation network
- `squeezesegv2Layers` — Create SqueezeSegV2 segmentation network

You can create a custom network layer-by-layer programmatically. For a list of supported layers and how to create them, see the “List of Deep Learning Layers” (Deep Learning Toolbox). To visualize the network architecture, use the `analyzeNetwork` function.

You can also design a custom network interactively by using the **Deep Network Designer**.

Train Network

To specify the training options, use the `trainingOptions` function. You can train the network by using the `trainNetwork` function.

Segment Point Clouds and Evaluate Results

Segment Point Cloud

Use the `pcsemanticseg` and `semanticseg` functions to obtain the segmentation results.

To segment buildings and vegetation from the aerial lidar data, use the `segmentAerialLidarBuildings` and `segmentAerialLidarVegetation` functions, respectively.

Evaluate Segmentation Results

Evaluate the segmentation results by using the `evaluateSemanticSegmentation` function.

Use Pretrained Segmentation Models

Lidar Toolbox provides these pretrained semantic segmentation models to perform point cloud segmentation and classification.

Pretrained Model	Description	Load Pretrained Model	Example
PointNet++	PointNet++ is a hierarchical neural network that captures local geometric features to improve the basic PointNet model. For more information, see “Getting Started with PointNet++”.	Load the pretrained model trained on DALES dataset: `load("pointnetplusTrained","net")`	“Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”

SqueezeSegV2	SqueezeSegV2 is a CNN for performing end-to-end semantic segmentation of an organized lidar point cloud. Training this network requires 2-D spherical projected images as inputs to the network.	Download the pretrained model trained on the PandaSet data set from Hesai and Scale: https://www.mathworks.com/supportfiles/lidar/data/trainedPointSegNet.mat	“Lidar Point Cloud Semantic Segmentation Using SqueezeSegV2 Deep Learning Network”
PointSeg	PointSeg is a CNN for performing end-to-end semantic segmentation of road objects based on an organized lidar point cloud. Training this network requires 2-D spherical projected images as inputs to the network.	Download the pretrained model trained on a highway scene data set from an Ouster OS1 sensor: https://www.mathworks.com/supportfiles/lidar/data/trainedPointSegNet.mat	“Lidar Point Cloud Semantic Segmentation Using PointSeg Deep Learning Network”

Code Generation

To learn how to generate CUDA® code for a segmentation workflow, see these examples.

- “Code Generation For Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”
- “Code Generation for Lidar Point Cloud Segmentation Network”

Note This functionality requires Deep Learning Toolbox™ licence.

See Also

Apps
Lidar Labeler | Deep Network Designer

Functions
`pointCloudInputLayer` | `squeezesegv2Layers` | `pointnetplusLayers` | `semanticseg` | `evaluateSemanticSegmentation`

Objects
`fileDatastore` | `pixelLabelDatastore` | `blockedPointCloudDatastore`

See Also

Related Examples

- “Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”
- “Lidar Point Cloud Semantic Segmentation Using PointSeg Deep Learning Network”

- “Lidar Point Cloud Semantic Segmentation Using SqueezeSegV2 Deep Learning Network”
- “Code Generation For Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”
- “Code Generation for Lidar Point Cloud Segmentation Network”

More About

- “Deep Learning in MATLAB” (Deep Learning Toolbox)
- “Getting Started with Point Clouds Using Deep Learning”
- “Getting Started with PointNet++”
- “Datastores for Deep Learning” (Deep Learning Toolbox)
- “List of Deep Learning Layers” (Deep Learning Toolbox)

Deep Learning with Point Clouds

Lidar sensors record point cloud data that provides rich 3-D geometric information of their surroundings. You can process this data to get a better understanding of an environment, and use it for various applications in driving, robotics, medicine, forestry, construction, urban planning, and oceanography.

Point cloud data is highly unordered and sparse as it stores points in a 3-D space without any discretion. Additionally, factors such as sensor range, occlusions, and uneven sampling of points also affect the nature of point cloud data. These factors make point cloud processing a challenging task.

Deep learning addresses various challenges in processing point cloud data. It is easier to perform complex point cloud processing tasks such as segmentation, detection, and tracking, by training deep learning networks.

- Segmentation clusters points in a point cloud and assigns class labels such as `car`, `tree`, and `building` to those clusters.
- Detection identifies and locates objects in a 3-D point cloud scene.
- Tracking tracks the state of objects across different point cloud frames.

These are the common steps for any point cloud processing workflow using deep learning.

Import Data

Lidar Toolbox provides functions to import and read raw point cloud data from several file formats. For more information, see “I/O”.

The toolbox enables you to divide this data into training and test data sets, and store them as datastore objects. For example, you can store point cloud files by using the `fileDatastore` object. For more information on datastore objects, see “Datastores for Deep Learning” (Deep Learning Toolbox).

The “Import Point Cloud Data For Deep Learning” example shows you how to import a large point cloud data set, and then configure and load a datastore.

Augment and Preprocess Data

In this step you prepare training data by labeling, augmenting, and preprocessing it.

Label Data

You need a large, labeled data set to train a deep learning network. If you have an unlabeled data set, you can use the **Lidar Labeler** app to label your training data. For information on how to use the app, see “Get Started with the Lidar Labeler”.

Augment Data

Data augmentation adds variety to the existing training data. The robustness of a network to data transformations increases when you train it on a data set with a lot of variety.

Augmentation techniques reduce overfitting problems and enable the network to better learn and infer features.

For more information on how to perform data augmentation on point clouds, see “Data Augmentations for Lidar Object Detection Using Deep Learning”.

Preprocess Data

You can preprocess your data before training the network. Lidar Toolbox provides function to perform various preprocessing tasks.

- Denoise, downsample, and filter point clouds.
- Convert unorganized data into the organized format.
- Divide aerial point cloud data into blocks to perform block-by-block processing.

For more information on preprocessing, see “Lidar Processing Applications” (Deep Learning Toolbox).

To interactively visualize, analyze, and preprocess point cloud data, use the **Lidar Viewer** app. For more information on how to use the app, see “Get Started with Lidar Viewer”.

Create Network

Every network has a unique architecture specific to the task you design it for. You define the network architecture based on the network input and the layers. Most deep learning networks either encode the point cloud into an image-like format, voxelize the point cloud, or operate directly on individual points.

Lidar Toolbox provides functions to create deep learning networks, as well as functions to use them for specific workflows.

- For segmentation workflows, see “Segmentation”.
- For object detection workflows, see “Detection and Tracking”.

To programmatically create a custom network layer-by-layer, use the functions specified in “List of Deep Learning Layers” (Deep Learning Toolbox). You can also interactively create the network by using the **Deep Network Designer** app.

To visualize the network structure, use the `analyzeNetwork` function.

Train Network

Specify training options by using the `trainingOptions` function, and train the network by using the `trainNetwork` function.

Test and Evaluate Results

Run the trained network on your test data set and evaluate the performance of the network. Depending on the task, you must use different functions to evaluate the results.

- To evaluate segmentation results, use the `evaluateSemanticSegmentation` function.
- To evaluate object detection results, use the `evaluateDetectionAOS` and `bboxOverlapRatio` functions.

Note This functionality requires Deep Learning Toolbox licence.

See Also

Apps
Lidar Labeler | **Deep Network Designer**

Functions
`pointCloudInputLayer` | `pointnetplusLayers` | `pointPillarsObjectDetector` | `trainPointPillarsObjectDetector`

Objects
`fileDatastore` | `pixelLabelDatastore` | `blockedPointCloudDatastore`

See Also

Related Examples

- “Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”
- “Code Generation For Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning”
- “Lidar 3-D Object Detection Using PointPillars Deep Learning”
- “Code Generation For Lidar Object Detection Using PointPillars Deep Learning”

More About

- “Deep Learning in MATLAB” (Deep Learning Toolbox)
- “Semantic Segmentation in Point Clouds Using Deep Learning”
- “Getting Started with PointNet++”
- “Getting Started with PointPillars”
- “Datastores for Deep Learning” (Deep Learning Toolbox)
- “List of Deep Learning Layers” (Deep Learning Toolbox)