

## Lab Guide

# LIDAR Point Cloud

## Content Description

The following document describes a LiDAR point cloud implementation in either python or MATLAB software environments.

Lab Guide	1
Running the example	2
Details	3
Running the Example	4
Details	4

## MATLAB

In this example, we will capture LIDAR data from the RP LIDAR A2 on the QCar platform, send the data to a polar plot, and generate a point cloud map. The process is shown in Figure 1.

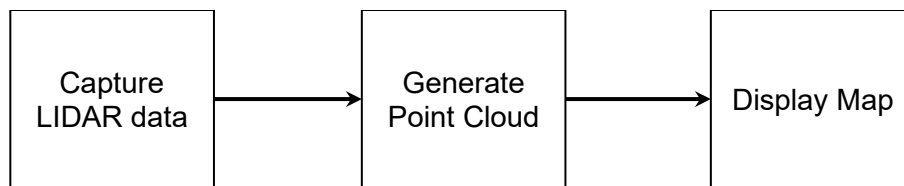


Figure 1. Component diagram

In addition, a timing module will be monitoring the entire application's performance. The Simulink implementation is displayed in Figure 2 below.

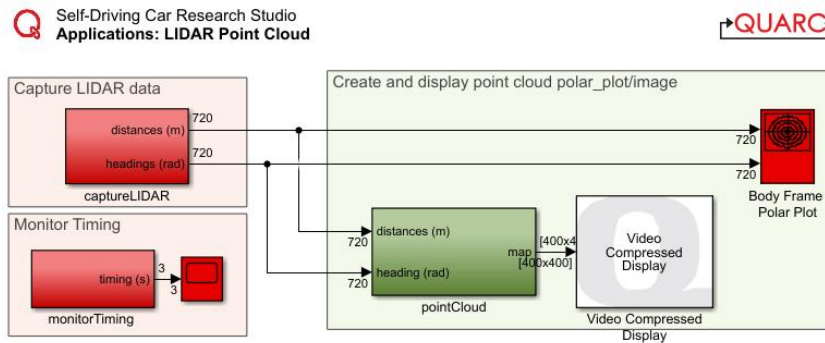
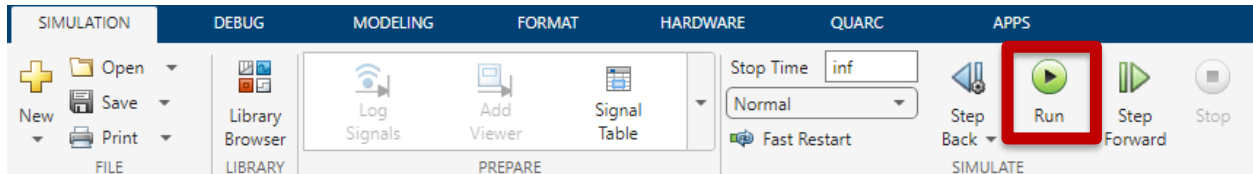


Figure 2. Simulink implementation of Lidar Point Cloud

## Running the example

To run examples for virtual QCar please go to the **SIMULATION** tab in the ribbon interface and click on the Run icon.



As your room size may vary, change the maximumDistance (m) parameter within the pointCloud subsystem accordingly, up to a maximum of 4m (corresponding to an 8 x 8 m room). Figure 3 shows the typical output expected when running this example.

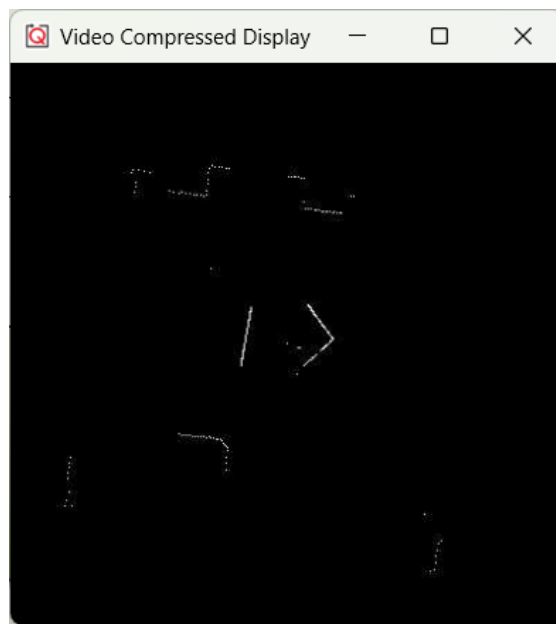
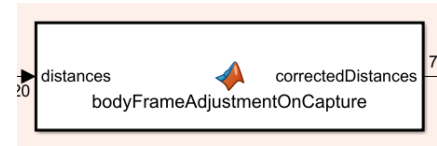


Figure 3. Point cloud map generated in a room.

## Details

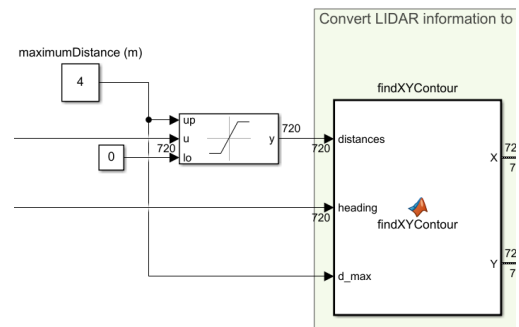
### 1. Capturing LIDAR data

The RP LIDAR A2 reads data in a clockwise manner, starting from a position opposite to the data cable attached to it. On the QCar platform, this corresponds to the +y axis. To correct this, the **captureLIDAR** subsystem corrects the order of the data to start at the front and orient counterclockwise to follow standard convention in the **bodyFrameAdjustmentOnCapture** function.



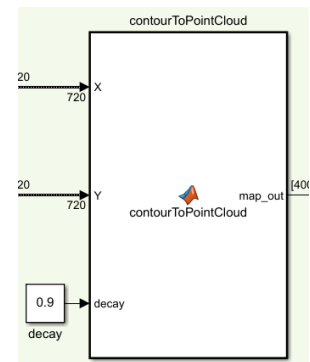
### 2. Saturating the distances data meaningfully

To limit the scope of the data to a range, the **distances** data is dynamically saturated using the **maximumDistance** parameter. The **findXYContour** function then converts the **distance/heading** data pairs to **X Y** pairs. However, we would like the **X Y** points corresponding to the **maximumDistance** to not show up within the point cloud itself, as they simply correspond to a maximum range and not physical obstacles. To do so, the **findXYContour** also drops data points that are equal to the **maximumDistance** parameter.



### 3. Generating the point cloud

This function first decays the existing map to 90%, thereby slowly erasing older data. The **X Y** data points in meters are converted to pixel scale **pX** and **pY** using a **gain** of 50 px/m for a map of size **dim** up to 400 pixels wide and tall (or 8m x 8m). Check out the documentation of MATLAB's **sub2ind** for information on how the (row, col) pairs in (**pX**, **pY**) are converted to indices where the point cloud map will be set to 1. Adjust the **decay** parameter to change the rate of update of the map. Note that you can do this online while the application is deployed.



### 4. Performance considerations

To improve performance, we only create a blank map on the first call by the use of persistent MATLAB variables. The variable **map\_internal** holds its value at any given iteration into the next call. At the end of the function, the **mapOut** is updated and then displayed.

## Python

In this example, we will capture LIDAR data from the RP LIDAR A2 on the QCar 2 and generate a point cloud map. The process is shown in Figure 4.

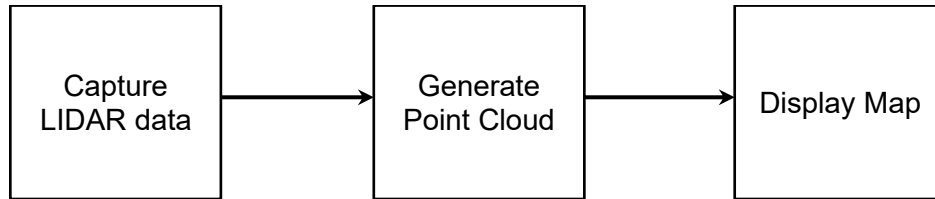


Figure 4. Component diagram

### Running the Example

Run the `lidar_point_cloud.py` example on your local machine. As your environment may vary in size, change the parameters **dim** and **gain** as you see fit. Figure 5 shows the typical output expected.



Figure 5. Point cloud map generated in a room.

### Details

#### 1. Capturing LIDAR data

The data available using the LIDAR class is already adjusted to be presented counterclockwise starting at the positive X axis of the lidar frame (right direction). The LIDAR class object has two attributes, **distances**, and **angles**, which correspond to

the data. The first line below shows the initialization step, and the second shows how to read the data.

```
myLidar = Lidar(  
    type='RPLidar',  
    numMeasurements=numMeasurements,  
    rangingDistanceMode=lidarMeasurementMode,  
    interpolationMode=lidarInterpolationMode)  
  
myLidar.read()
```

## 2. Converting distances/angles to x y

After heading angles are converted from lidar frame to QCar body frame, the **distance/heading** data pairs are converted to **x y** pairs (in meters) using the lines below, and then to **pX, pY** pairs (in pixels) for the image.

```
x = myLidar.distances[idx]*np.cos(anglesInBodyFrame[idx])  
y = myLidar.distances[idx]*np.sin(anglesInBodyFrame[idx])  
  
pX = (sideLengthScale/2 - x*pixelsPerMeter).astype(np.uint16)  
pY = (sideLengthScale/2 - y*pixelsPerMeter).astype(np.uint16)
```

## 3. Generating the point cloud

Note that the **map** is set to zeros at the beginning.

```
map = np.zeros((dim, dim), dtype=np.float32)
```

It then decays slowly based on the **decay** parameter at the start of the loop.

```
map = decay*map
```

A line below updates the **map** at the locations **pX, pY** near the end of the loop.

```
map[pX, pY] = 1
```

## 4. Performance considerations

To improve performance, we only create a blank map when initializing the code. Within the main loop, older map data slowly decays. The module **opencv** provides the **waitKey()** method for pausing in this case. See [User Manual – Software Python](#) for more information on timing.