Velodyne Lidar Packet Decoding and Visualization[⋆,⋆⋆]

Sanam Shakya[a,1,], second author[a,1,1]

[a]*LogicTronix, Kupondole, Lalitpur, Nepal 40011*

Abstract

This document is for interfacing Velodyne VLP16 Lidar to Linux system. VLP16 Lidar is interfaced with ethernet to system and sends data through UDP port. So data packets are captured from UDP socket, then it is decoded into (x, y, z, r) Lidar dataframes. After decoding the data, top view visualization is done using OpenCV library.

Keywords: lidar, lidar decoder, Velodyne VLP16

1. Introduction

Velodyne VLP-16 Lidar sends data stream through UDP ethernet port to the connected system. For capturing UDP packets asynchronous C++ Boost socket library is used. Captured packets consists of Data Packets and Position Packets corresponding to 360° scan. For further processing and visualization captured packets are decoded into custom Lidar point data structure (x, y, z, r) and Lidar point frames for a 360°scan. For visualizing the Lidar point frames, top view is used so only (x, y) from Lidar point is used.

⋆This document is a collaborative effort.

⋆⋆The second title footnote which is a longer longer than the first one and with an intention to fill in up more than one line while formatting.

Email address: sanam@logictronix.com (Sanam Shakya)

URL: http://www.logictronix.com (second author)

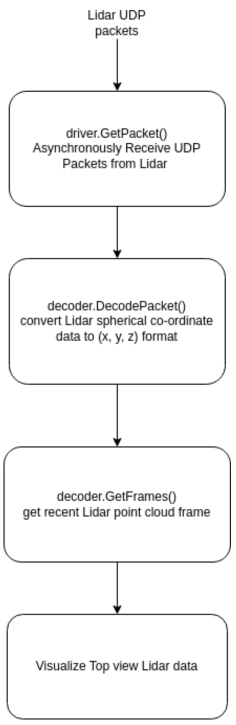

Lidar UDP packets
driver.GetPacket() Asynchronously Receive UDP Packets from Lidar
decoder.DecodePacket() convert Lidar spherical co-ordinate data to (x, y, z) format
decoder.GetFrames() get recent Lidar point cloud frame
Visualize Top view Lidar data

Figure 1: Code flow

2. Lidar setup

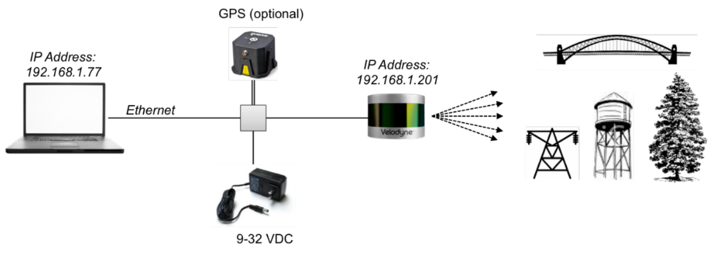


Figure 2: Velodyne Lidar setup

Velodyne Lidar Specs (Detail refer to Velodyne [2]) :

- Field of view(vertical) : $30^\circ (+15^\circ to - 15^\circ)$
- Angular resolution (vertical) : 2°
- Field of View (horizontal/azimuth) : 360°
- Angular resolution (horizontal) : $0.1^\circ - 0.4^\circ$
- 16 channels
- Up to 0.3 million points/second
- 100 Mbps Ethernet connection

3. Velodyne VLP-16 working

Velodyne VLP16 consists of 16 laser and detector in compact housing. The laser fires thousands of time and time of flight of reflected light is processed to obtain the distance of obstruction in the environment. Further laser-detector housing spins full 360° giving horizontal angle - azimuth (α) and distance to the object. Vertical or elevation angle (ω) is fixed varying from +15° to -15° with 2° spacing and given by the Laser ID ranging from 0-31. The horizontal or azimuth (α) angle is reported at start of each firing sequence[1]. The information from two Firing Sequences of 16 lasers is contained in one Data Block, consisting of 32 data records. Each packet contains the data from 12*2=24 firing sequences. Only one Azimuth is returned per Data Block.

4. VLP-16 Data Packet Format and Decoder Code Flow

VLP-16 data packet is 1248 bytes long, consisting 42 byte UDP header and 1206 byte payload containing 12 blocks of 100 byte data records, followed by a 4 byte time stamp and 2 factory bytes. Each data block begins with a two byte start identifier "FF EE", then a two-byte azimuth value (α), followed by 32 x 3 byte data records. More information on VLP-16 data packet is available at VLP-16 user guide Velodyne [1]. Graphical representation of data frame and data records is show in Figure:3 and Figure:4.

A single 1248 bytes data packet consists of 12 * 2 firing sequence

Azimuth angle is stored in two bytes ranging from 0.00° to 359.99°.

While distance is reported with resolution of 2.0 mm. So to calculate the absolute distance, it has to be multiplied by 2.0 mm or 0.002 m

For storing Lidar data packet following data structure is used:

```
typedef struct HDLLaserReturn
{
    unsigned short distance;
    unsigned char intensity;
} HDLLaserReturn;
struct HDLFiringData
{
    unsigned short blockIdentifier;
    unsigned short rotationalPosition;
    HDLLaserReturn laserReturns[HDL_LASER_PER_FIRING];
};
struct HDLDataPacket
{
    HDLFiringData firingData[HDL_FIRING_PER_PKT];
    unsigned int gpsTimestamp;
    unsigned char blank1;
    unsigned char blank2;
};
```

In decoder code, recieved data packet array is converted to *HDLDataPacket* format through casting.

```
HDLDataPacket* dataPacket = reinterpret_cast<HDLDataPacket *>(data);
```

[1]Firing Sequence : The time and/or process of cycle-firing all the lasers in a VLP-16. It takes 55.296 uSec to fire all 16 Lasers.

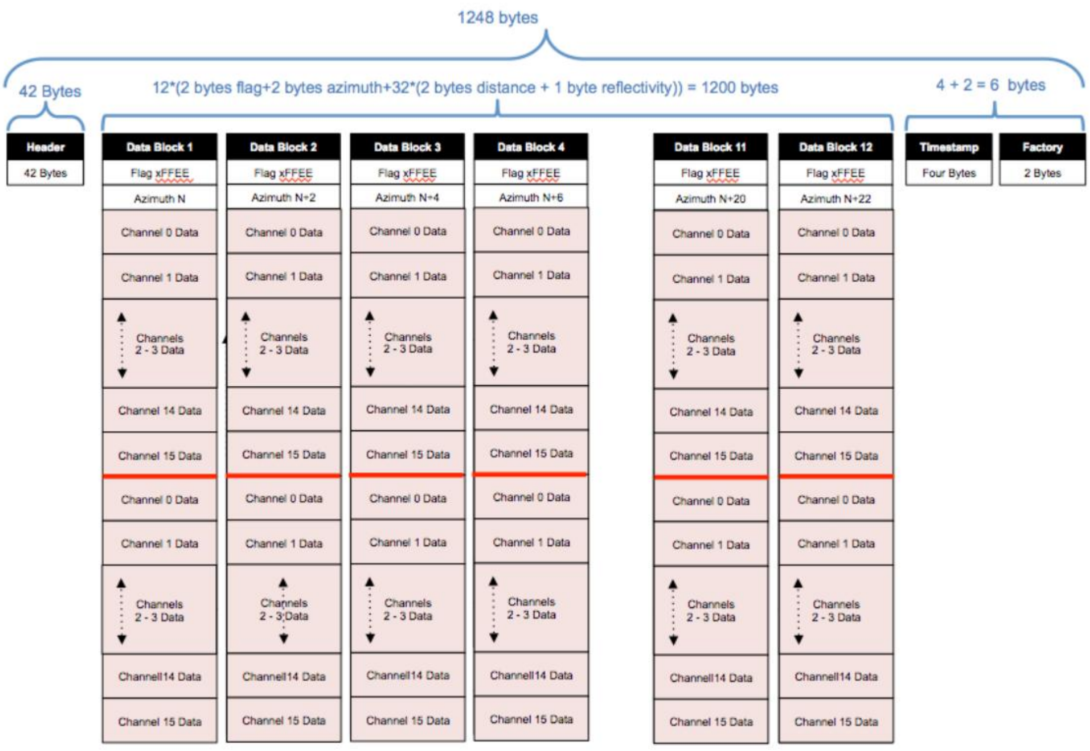


Figure 3: Data Packet

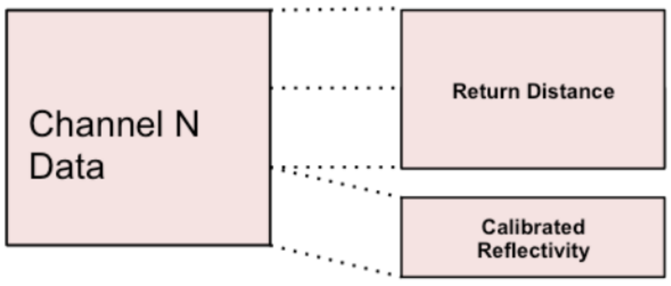


Figure 4: Data Record Velodyne [1]

4.1. Lidar frame generation

For frame generation, data packet from 0° to 360° is made into single frame by comparing consecutive azimuth angle (*rotationalPosition)*. New frame is started as azimuth angle becomes smaller than its consecutive angle i.e. when angle change from 360° to 0°. Corresponding code in *PacketDecoder.cpp* code in implementation of *ProcessHDLPacket()* function:

```
void PacketDecoder::ProcessHDLPacket(unsigned char *data, unsigned int
    data_length)
{
  if (data_length != 1206) {
    std::cout << "PacketDecoder: Warning, data packet is not 1206 bytes" <<
        std::endl;
    return;
  }

  HDLDataPacket* dataPacket = reinterpret_cast<HDLDataPacket *>(data);

  for (int i = 0; i < HDL_FIRING_PER_PKT; ++i) {
    HDLFiringData firingData = dataPacket->firingData[i];
    int offset = (firingData.blockIdentifier == BLOCK_0_TO_31) ? 0 : 32;
    // std::cout << "offset value : " << offset << std::endl;

    if (firingData.rotationalPosition < _last_azimuth) {
      SplitFrame();
    }

    _last_azimuth = firingData.rotationalPosition;

    for (int j = 0; j < HDL_LASER_PER_FIRING; j++) {
      unsigned char laserId = static_cast<unsigned char>(j + offset);
      if (firingData.laserReturns[j].distance != 0.0) {
        PushFiringData(laserId, firingData.rotationalPosition, dataPacket->
            gpsTimestamp, firingData.laserReturns[j], laser_corrections_[j +
            offset]);
      }
    }
  }
}
```

Next structured Lidar data frame is passed to *PushFiringData()* function which converts distance and azimuth angle data into (x, y, z, r) - LidarPoint data.

4.2. Lidar Spherical co-ordinate data to 3D Cartesian co-ordinate conversion

Spherical co-ordinate obtained for Lidar single channel (R, α,ω) is converted to 3D (X, Y, Z) cartesian co-ordinates using following formula and shown in Figure : 5

$xyDistance = R * cos(\omega)$
$X = R * cos(\omega) * sin(\alpha)$
$Y = R * cos(\omega) * cos(\alpha)$
$Z = R * sin(\omega)$

In code above conversion formula is implemented as follows:

- In packet decoder class constructor, *InitTables()* function is called which creates cos and sine lookup table from (0.00° to 360.01°)

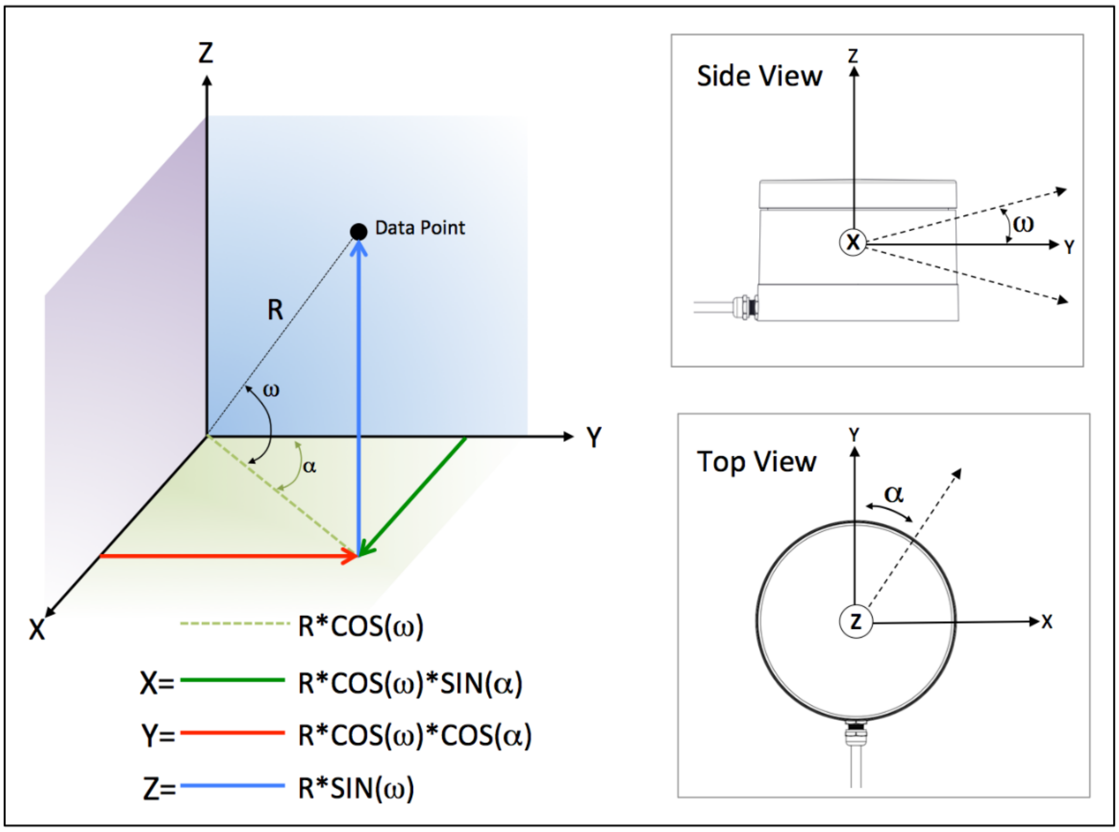


Figure 5: Spherical to 3D Cartesian conversion

- Next sine and cosine of elevation angle corresponding to each Laser ID is calculated and stored in *laser_ corrections_ [laser_ id].sinVertCorrection* and *laser_ corrections_ [laser_ id].cosVertCorrection*
- After creation of structured *HDLDataPacket* dataPacket* in *ProcessesHDLPacket()*, laser ID is created and pass each data record : *firingData.laserReturns[laser_ id]* to *PushFiringData()* function which converts individual laser data record to XYZ data and pushed to *LindarPoint* vector array.
- In *PushFiringData(),*
  - First cosine and sine of azimuth angle - *cosAzimuth* and *sinAzimuth* is calculated using lookup table.

    ```
    cosAzimuth = cos_lookup_table_[azimuth];
    sinAzimuth = sin_lookup_table_[azimuth];
    ```
  - Next *xyDistance* is calculated which is used to calculate X and Y co-ordinates

    ```
    double xyDistance = distanceM * correction.cosVertCorrection -
        correction.sinVertOffsetCorrection;
    double x = (xyDistance * sinAzimuth - correction.
        horizontalOffsetCorrection * cosAzimuth);
    double y = (xyDistance * cosAzimuth + correction.
        horizontalOffsetCorrection * sinAzimuth);
    ```
  - For calculating Z co-ordinate first absolute distance is calculated and multiplied with *correction.sinVertCorrection*

    ```
    double distanceM = laserReturn.distance * 0.002 + correction.
        distanceCorrection;
    double z = (distanceM * correction.sinVertCorrection + correction.
        cosVertOffsetCorrection);
    ```
  - Next obtained (x, y, z) co-ordinate and reflectivity (r) is pushed into *lidar_ points* vector array

Code snippet of *PushFiringData():*

```
void PacketDecoder::PushFiringData(unsigned char laserId, unsigned short
    azimuth, unsigned int timestamp, HDLLaserReturn laserReturn,
    HDLLaserCorrection correction)
{
  double cosAzimuth, sinAzimuth;
  LidarPoint lidar_point;
  if (correction.azimuthCorrection == 0) {
    cosAzimuth = cos_lookup_table_[azimuth];
    sinAzimuth = sin_lookup_table_[azimuth];
    // std::cout << "using table based sin and cos" << std::endl;
  } else {
    double azimuthInRadians = HDL_Grabber_toRadians((static_cast<double> (
        azimuth) / 100.0) - correction.azimuthCorrection);
    cosAzimuth = std::cos(azimuthInRadians);
    sinAzimuth = std::sin(azimuthInRadians);
  }

  double distanceM = laserReturn.distance * 0.002 + correction.
      distanceCorrection;
```

```
  double xyDistance = distanceM * correction.cosVertCorrection - correction.
      sinVertOffsetCorrection;

  double x = (xyDistance * sinAzimuth - correction.horizontalOffsetCorrection
      * cosAzimuth);
  double y = (xyDistance * cosAzimuth + correction.horizontalOffsetCorrection
      * sinAzimuth);
  double z = (distanceM * correction.sinVertCorrection + correction.
      cosVertOffsetCorrection);
  unsigned char intensity = laserReturn.intensity;
  lidar_point.x = x ;
  lidar_point.y = y ;
  lidar_point.z = z ;
  lidar_point.r = (double) intensity ;


  _frame->x.push_back(x);
  _frame->y.push_back(y);
  _frame->z.push_back(z);
  _frame->intensity.push_back(intensity);
  _frame->laser_id.push_back(laserId);
  _frame->azimuth.push_back(azimuth);
  _frame->distance.push_back(distanceM);
  _frame->ms_from_top_of_hour.push_back(timestamp);
  _frame->lidar_points.push_back(lidar_point);
}
```

5. Top View Visualization of Lidar *3D* data:

For visualizing the Lidar data from Top View, a 2D image is created using OpenCV frame. Each point (X, Y, Z, r) data of *lidar_points* are iterated and a point circle is drawn corresponding to (x, y) co-ordinates. Code explanation: Top view visualization is implemented in *showLidarTopView()* function.

- First image size (*imageSize)* and world size (*worldSize)* is set and *cv::Mat topviewImg* OpenCV Mat frame is created.
- Next each point in *lidar_points* are iterated and converted to image co-ordinate. For this (X, Y) world co-ordinate obtained from Lidar data are converted to pixel co-ordinate.
- $x_{image} = X_{world} * \frac{imageSize.height}{worldSize.height}$
- $y_{image} = Y_{world} * \frac{imageSize.width}{worldSize.width}$
- Then offset is added with image origin shifted to center of image given by $(\frac{imageSize.height}{2}, \frac{imageSize.width}{2})$
- Now for obtained x_{image}, y_{image} pixel co-ordinates a circle is created with center point(x_{image}, y_{image})

6. Building and Testing the Lidar Viewer application

- Get the source from github: https://github.com/LogicTronixInc/Velodyne-Lidar-Data-Processing-and-Visualization

- In src folder create the build directory

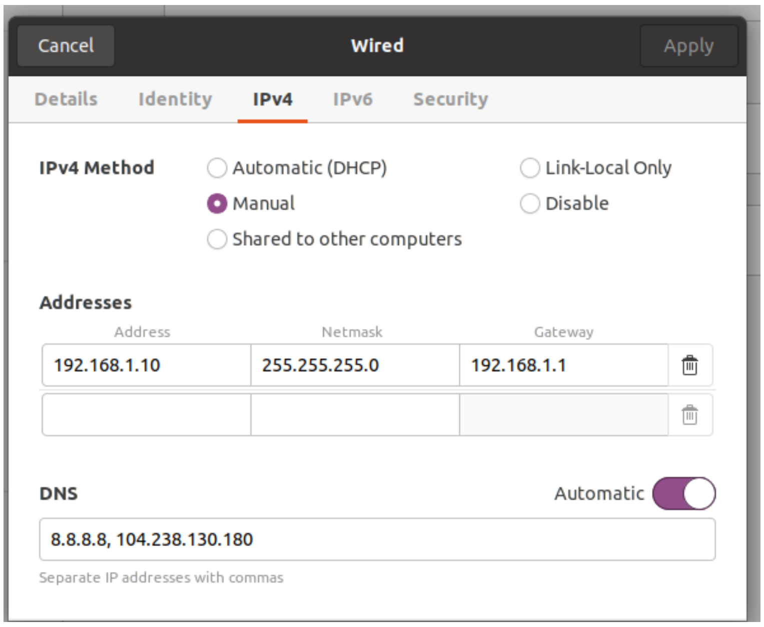
```
mkdir build
```

- Change to build directory and run cmake and then run make

```
cd build
cmake ..
make
```

- This will create *test_ PacketDecoder* app along with libraries.

- Now connect the Lidar to the host machine through ethernet and setup the host PC network to same domain as Lidar.



IPv4 Method: ( ) Automatic (DHCP) [✓ Manual] ( ) Shared to other computers ( ) Link-Local Only ( ) Disable

Addresses

Address	Netmask	Gateway
192.168.1.10	255.255.255.0	192.168.1.1

DNS Automatic [✓ on]

8.8.8.8, 104.238.130.180

Separate IP addresses with commas

Figure 6: Host IP settings

- Now run the *test_ PacketDecoder* app to view the Lidar data

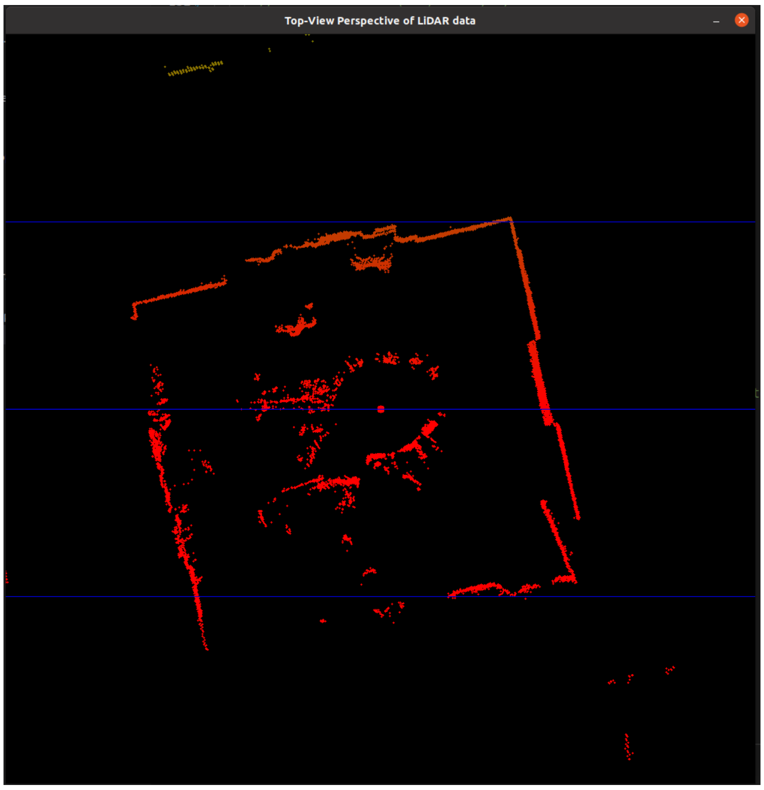


Figure 7: test_PacketDecoder app output graphics

References

[1] Velodyne, 2015. Vlp-16 user manual.
[2] Velodyne, 2024. Vlp-16 product guide. URL: `www.velodynelidar.com`.