towards
data science

Follow 525K Followers

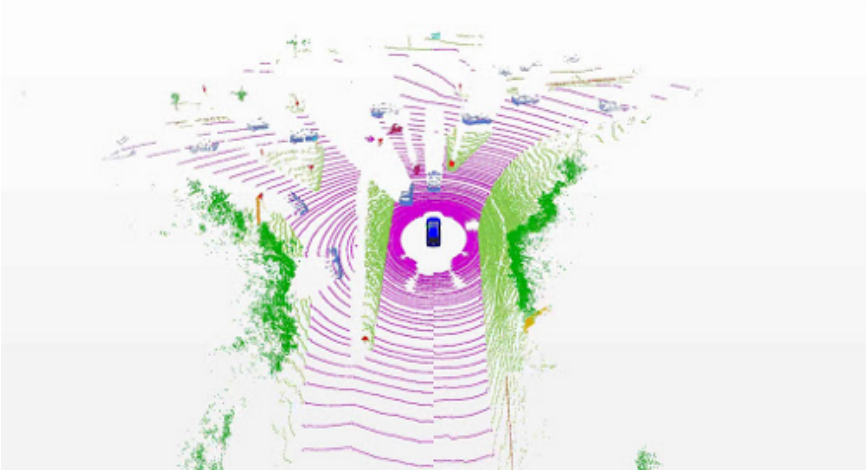
Spherical Projection for Point Clouds

Anirudh Topiwala Mar 28, 2020 · 8 min read

Hello Everyone, this is my first medium post and I hope to engage everyone till the very end. In this post, I am going to talk about how to project a 3D point cloud into an image using spherical projection. I have also written code for the same which is available on my GitHub repository here. So lets get started!!!

Spherical Projection or Front View projection is nothing but a way to represent the 3D point cloud data into 2D image data, and so essentially, it also acts as a

dimensionality reduction method. Spherical projection is increasingly being used in different deep learning solutions for processing point clouds.



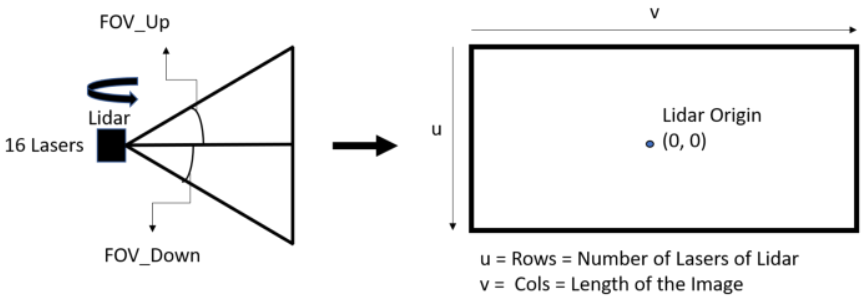
Semantic Segmentation (Author: SemanticKitti)

One of the most applied area is to classify and segment objects in a point cloud. This can be seen in various publications like PointSeg, SqueezeSeg, SalsaNet and so on. The biggest advantage of representing a point cloud as an image is that it opens up all the research done in the last decade on 2D images to 3D point clouds. For example, different state-of-the-art networks like FCN, U-Net, Mask-RCNN, and Faster-RCNN can now be extended to point cloud data and this is what a lot of

researchers are trying to accomplish, especially in the self-driving field.

Intuitive Explanation

To understand this lets dig into how one point cloud scan is formed by a lidar. Let's consider the case of a 16 laser lidar shown in the figure below.



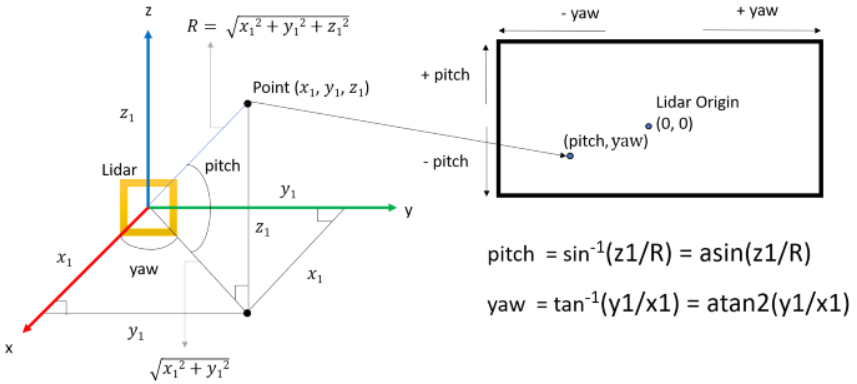
The figure on the left shows a lidar which consists of 16 lasers. The max and min angle of view formed by the first and the last laser of the lidar are indicated by FOV_Up and FOV_Down. The rectangle on the right is what we get when we project the hollow cylinder formed by the lidar onto a plane. (Author: Anirudh Topiwala)

Each of the 16 lasers are oriented at a fixed angle depending upon the vertical angular resolution, FOV_Up (Upper Field of View) and FOV_Down (Lower Field of View). Each lidar has a firing and a receiving unit. The points are formed by calculating the time of flight for each laser after it has been reflected from the object.

These 16 lasers when rotated by 360 degrees form one point cloud.

> *Thus the geometric shape this will result into is a **hollow cylinder** with the lidar at its center. When we project this hollow cylinder looking from an axis perpendicular to principal axis of the cylinder, onto a plane, we get an image. This image is called spherical projection image.*

Let's get technical with some Math now!!



(Author: Anirudh Topiwala)

Our goal is to find pixel coordinates of the projection image for all (x,y,z) points. This can be achieved by using the spherical coordinate system. Looking at the above figure, we can see that if the +ive x-axis is the front view of the lidar, then each point of the cloud will

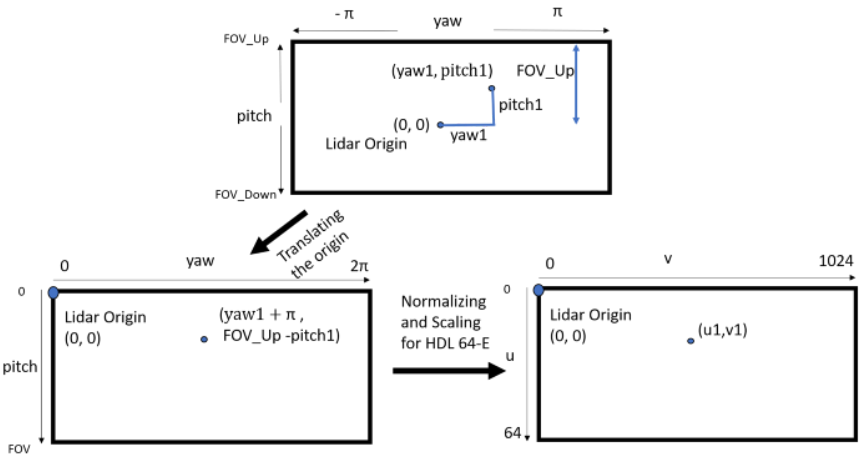
make an inclination *angle*, ***pitch*** with the xy plane and *angle*, ***yaw*** with the xz axis. Here we want to project the hollow cylinder onto the zy plane to form the image. By employing basic trigonometry, we can get the value of pitch and yaw for each point. As the origin is at the center of the image, these yaw and pitch values make up each pixel position of the projection image. Thus, by calculating the yaw and pitch for each point, we can completely form the projection image. At this point I would also like to point out, the value of pitch will range from [FOV_Up , FOV_Down] as it is max and min angle pointed by the lasers and the value of yaw will range from [-pi, pi] as that is the range given by atan2 function.

$$\text{pitch} = \sin^{-1}(z1/R) = \text{asin}(z1/R)$$

$$\text{yaw} = \tan^{-1}(y1/x1) = \text{atan2}(y1/x1)$$

Although we now have a projection image, we still cannot use it due to two problems. First, we need to translate the origin to the top left corner of the image as shown below. This is because, in computer vision, the

norm is to have the origin at the top left corner of the image. Second, we need to scale the image depending upon the type of lidar being used. Both of these steps are shown below:



(Author: Anirudh Topiwala)

1. Translating the Origin

This is a simple translation of origin problem. Looking at the above figure, we can move the origin to the left edge of the image by adding an angle of pi to the yaw axis, whereas the origin is moved to the top of the image by subtracting pitch angle from FOV_Up. Thus the new equations become:

$$pitch = FOV\_Up - pitch$$

$$yaw = yaw + \pi$$

2. Normalizing and Scaling

> *This step is necessary as size of the projection image will vary for different types of lidar with the prime objective being to fit as many points as possible in the projection image.*

As we have observed throughout this blog post that this whole process is about dimensionality reduction and therefore, its a fact that we will loose data. The idea here is to form the image dimensions in such a way that we are able to capture the most relevant points from the cloud in the image. Therefore, we can logically see that the **number of lasers of the lidar should be equivalent to the width of the projection image**. This way each row of the image will relate to points obtained from each laser of the lidar. This value is called the row_scale factor and is multiplied to the normalized pitch axis. For example, a Velodyne HDL 64-E lidar has 64 lasers and therefore, the width of the projection image should be set to 64 by multiplying this value to the pitch axis.

The length of the image makes up the angular resolution in yaw. This parameter can be tuned to fit as many points as possible. This length of the image is also called the col_scale and is multiplied to the normalized yaw axis to get the final image dimensions. Again, in the case of HDL 64-E, the max horizontal resolution is 0.35 degrees. Therefore in the worst case, we will at least get (360/0.35 = 1028) points per laser. In deep learning and for convolution networks, image sizes are preferred to be multiples of 2. Therefore, a length of 1024 would fit maximum number of points in the image and we will multiply this value to yaw axis.

Therefore, the image dimensions of 64X1024 are ideal for HDL 64-E. A similar calculation can be made for Velodyne VLP-16 where the best image dimensions come out to be 16x1024.

The normalizing equation can now be rewritten as:

$$\text{normalized\_pitch} = (\text{FOV\_Up} - \text{pitch}) / (\text{FOV\_Up} - \text{FOV\_Down})$$

$$\text{normalized\_yaw} = (\text{yaw} + \pi) / 2\pi$$

if the overall FOV = FOV_Up + abs(FOV_Down), then the final equations for (u,v) with some rearrangement of

values can be written as:

$$u = pitch = row\_scale * (1 - (pitch + FOV\_down) / FOV)$$
$$v = yaw = col\_scale * (0.5 * ((yaw / \pi) + 1))$$

and for Velodyne HDL 64-E lidar, the same equations can be rewritten as:

$$u = pitch = 64 * (1 - (pitch + FOV\_down) / FOV)$$
$$v = yaw = 1024 * (0.5 * ((yaw / \pi) + 1))$$

> *Thus, with the help of the above equations, we can project each point (x,y,z) of the cloud into its corresponding spherical projection (u,v).*

Encoding Point Information into the Image

Once we have the (u,v) pixel for each point of the cloud, we need to round it to get the nearest integer and encode point information into it. Usually, 5 key values are added at each pixel and that is X, Y, Z , R and I. Here, (X, Y, Z) are the coordinates of the point, R is the euclidean distance or range of that point from the lidar and I is the intensity. Range is the only value that we calculate, as X,Y,Z and I values are already present for

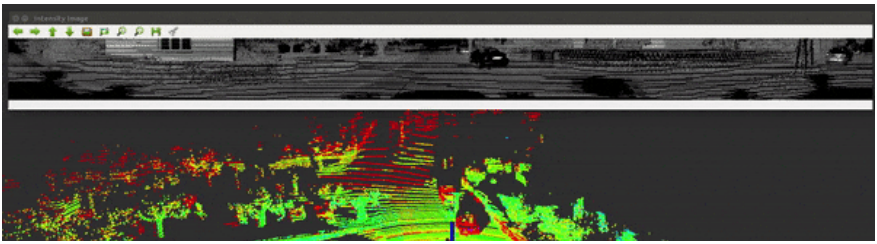
each point of the cloud formed by the lidar. Therefore, the final image dimensions we get are:

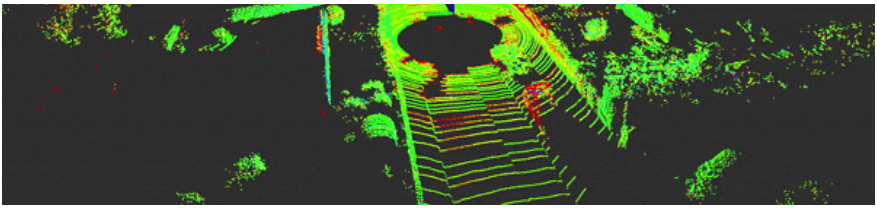
> ***dimension of image = Num of lasers of lidar * Length of the projection image * 5 (the five channels are X,Y,Z,R,I)***

Now the Part you have been waiting for.. Results!!!

Using the equations derived above, I have written a c++ code which can be found here. This code uses PCL library to load the point cloud and openCV to visualize each dimension of the Spherical Image formed. To test my implementation, I have used dataset from SemanticKitti. The upper part of the gif below shows the intensity values for the spherical image formed from the point cloud shown in the lower part.

> *To recap, as the lidar used in this dataset is velodyne HDL 64-E, the dimensions of the Spherical image formed is 64 * 1024* 5.*





The Upper part is the intensity dimension of the Spherical Image formed and the lower part is the input point cloud. (Author: Anirudh Topiwala)

Some key observations:

> *1. Each row of the spherical image corresponds to points obtained from each laser of the lidar. Here, the lowest row in the image corresponds to the lowest laser in lidar which is the closest ring near the lidar as seen in the gif above.*
>
> *2. The spherical image formed is circular in nature, meaning you would be able to observe that when objects leave the image from the left side, they might reappear on the right side of the image.*
>
> *3. The objects in the image can be identified with the human eye. Meaning, if you look at the image, it is easy to distinguish cars, cycles, buildings, road and so on. And if it is distinguishable with the human eye, then there is a really good chance that a deep network trained on such images is able to give good segmentation and classification results.*

> *4. The points lost in this projection are those which have the same pitch and yaw angles with the lidar center but different range values. So, only one of the points on the ray connecting the lidar center to that point would be captured in the projection image.*

A clear video of the result can be found here.

To conclude, spherical projection is a very powerful tool to represent point cloud data in image form. Using it can help us extend all the research done over the last decade in image space to point cloud data and therefore has varied applications in the realms of Robotics and Autonomous vehicles.

I am open to any suggestions or brain storming sessions for this topic as well as other topics in the realms of computer vision and deep learning, so please feel free to contact me!!!

Spherical Projection | Point Cloud | Deep Learning | Semantic Segmentation