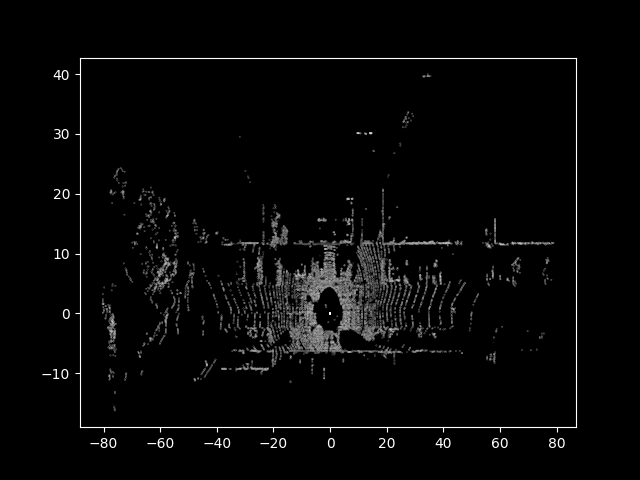
## Problem 1

**Description**

The data from the dictionary is loaded and the required velodyne data is obtained from this. The x, y and the alpha(reflectance) values of all the points are required for creating the BEV image. To create the required resolution and max pooling the following steps were performed.

1. Find the number of pixels with the given resolution and the current image dimensions in both axes.
2. Create a temporary matrix with the number of rows equal to the total number of pixels and three columns for the x, y and alpha values. Now we need to do max pooling for this array and given resolution.
3. Find the grid location for each of the points in the point cloud by normalizing them with the given resolution on both x and y coordinates and then store them in the temporary matrix if the reflectance value of the current coordinate is more than the reflectance value of any other point which might already be in the same grid location. This is done by looping all the points in the point cloud.
4. The reflectance value is initially set to 0 for all the points in the temporary matrix so there will be no problem for a point to go into the grid location if there were no points already there.
5. All the rows with no data are then filtered out to finally form the 2D point array with max pooling performed.

The plot is then created using scatterplot with black background as requested. The cmap option of Greys is being used.

**Result**  
  
 

## Problem 2

**Description** The dictionary is loaded and the required data is loaded from this. For this problem the velodyne point cloud, cam2 image, the intrinsic and extrinsic matrices, semantic labels, colour map, cam0 rotation, bounding box coordinates and dimensions were required. For the first subtask the following steps were performed.

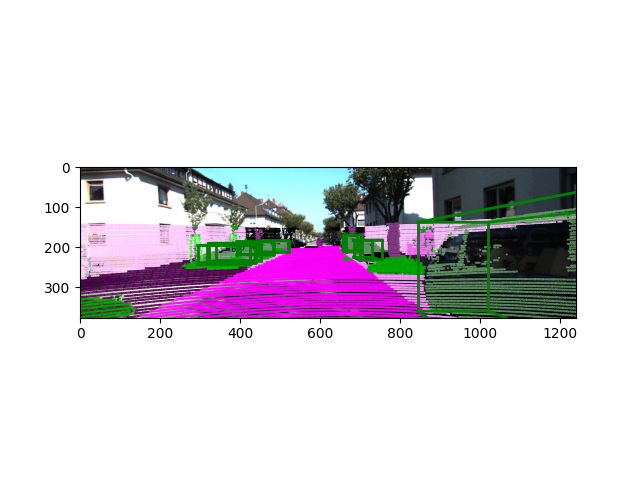
1. The velodyne data is filtered by removing all the points with negative x coordinates and then the respective color maps are linked to each of the points with the use of the corresponding semantic labels.
2. An extra column was added to the filtered velodyne data with values of 1 to create a matrix with 4 columns to be used in the conversions of the coordinates to the camera frames
3. The point cloud is then transformed to cam0 coordinates by using the T\_Cam0\_velo conversion matrix.
4. The resulting matrix with points in cam0 coordinates is then projected to the cam2 coordinates by using the P\_rect\_20 matrix. This matrix was normalized with the 3rd column to get the 2D coordinates. Now we have the point cloud projected to cam2 coordinates and the corresponding color maps.
5. Also as the color map provided is in BGR format and as the plotter used RGB format this rearrangement was made to the color map.

**Result**

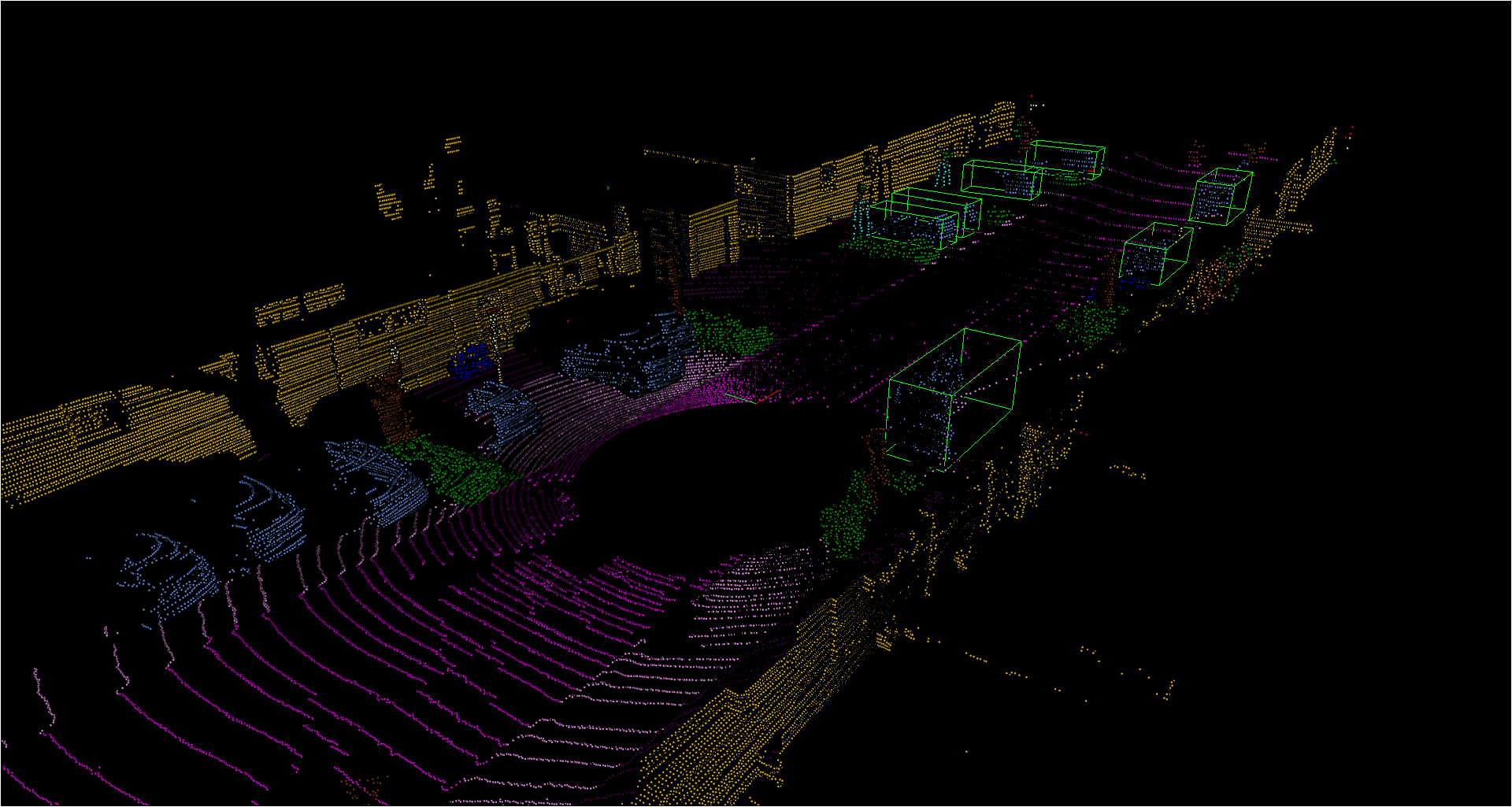
The following additional steps were performed to obtain and project the bounding boxes to the above image.

1. For each bounding box list in the dictionary the rotation matrix was created based on the given coordinate vector for the camera 0. This was essentially a rotation in coordinates about the y axis.
2. The respective coordinates of the mid point of the bottom face of the bounding boxes were stored along with the dimensions of the bounding box.
3. The rotation correction was then applied to the given dimensions of each bounding box and then was used to shift the given centre point of the bottom face to the respective edges. The points were then projected from the cam0 coordinates to cam2 coordinates using the P\_rect\_20 matrix. The order of the edges was maintained as given in the 3dvis.py code to make it easier to use the same connect matrix.
4. As we did not use vispy for the first 2 subtasks of this problem a loop was created and a line was plotted between the edges of the bounding box with the use of the connect matrix coordinates.
5. Each line was plotted in the same image forming the bounding boxes perfectly.

**Result**

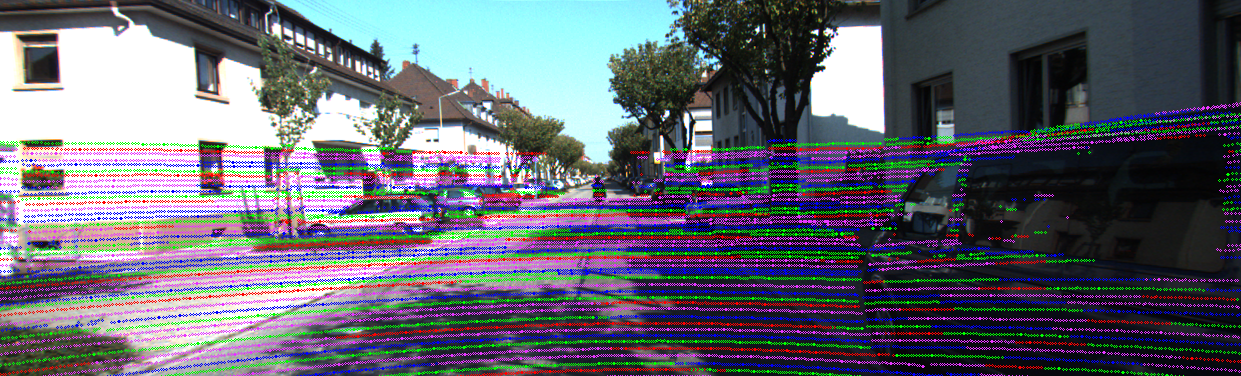
For the visualization in 3 dimensions the following steps were performed in the given 3dvis.py code.

1. As we use the velodyne data directly here, the color map is linked to the corresponding points in the poin cloud with the use of the semantic labels. No filtering is done here as before and all the points are linked to their respective color maps.
2. The conversion from BGR to RGB is done again to be used by the vispy color map reader.
3. The scene is plotted with the color map included inside this function.
4. Now to include the bounding boxes in the scene we just need to pass all the edges of each of the bounding boxes in the same order as given in the code(or the connect matrix) to the update boxes function.
5. The same steps were performed to correct the rotation and to obtain the corrected coordinates of the boxes in the cam0 coordinates.
6. To convert these coordinates to 3D points we take the dot product of these point array with the inverse of the T\_cam0\_velo matrix. Now we have the required array containing the edge coordinates of the bounding boxes.
7. This is passed to the update\_boxes function to plot the bounding boxes in the same scene

**Result**

From this 3D visualization we can see that our network did not capture one car in between the 3rd and 4th car captured to the left side in the surroundings and also one more car just after the last captured car in the right side which lie within the camera range.

## Problem 3



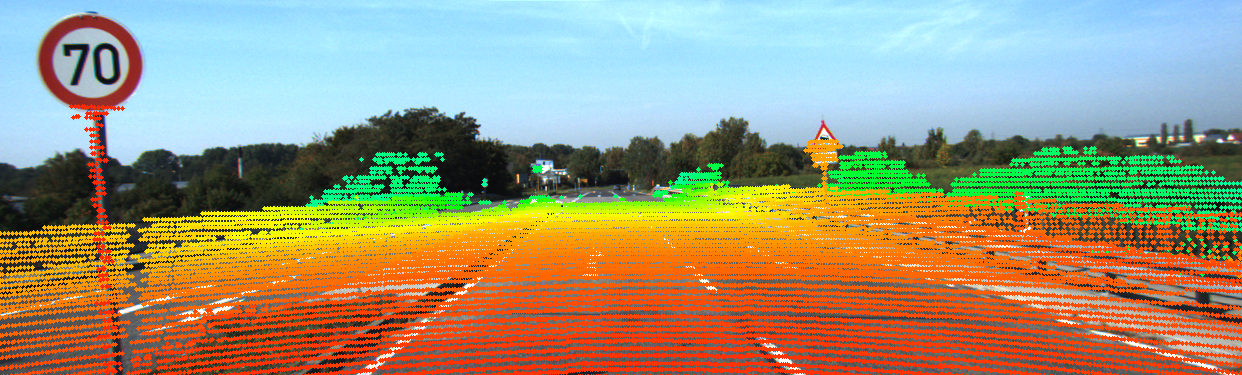
The dictionary with the required data is loaded. For this problem the velodyne point cloud, cam2 image, the intrinsic and extrinsic matrices and velodyne data sheet information was required.

1. Extract the vertical angle (inclination) in Lidar frame
   1. Calculate the distance r of the point (euclidean norm) and use arctan2 with z coordinate
2. Group into different channels
   1. From datasheet we know the FoV in vertical direction, as well as the number of channels
   2. We can divide the FoV (vertical) by the number of channels to get angular separation between the channels. Then we segment the FoV in vertical direction into the individual channels
   3. Then assign the points using the angle extracted in 1. to the nearest channel from 2.b.
3. In order to visualize the results, the Lidar points were transformed from the Lidar frame into the camera frame
   1. Transforme the velodyne point cloud into homogeneous coordinates
   2. Use the t\_cam2\_velo matrix to transform to camera frame
   3. Remove all points with z (forward) smaller 0 to be left only with points in camera frame
   4. Project from the 3d homogeneous camera plane onto the 2d homogeneous image plane using P\_rect\_20 matrix and normalize the result over the homogeneous coordinate
   5. Transform into 3d, by dropping 4th dimension
4. The channels where split into 4 alternating groups in order to tell apart the channels on the image

## Problem 4

The dictionary with the required data is loaded. For this problem the velodyne point cloud, cam2 image, the intrinsic and extrinsic matrices, imu2velo and imu data was required.

1. Calculate the angle in the xy-plane of each point relative to camera (or front)
   1. Use the arctan2 function on the (x,y)
   2. The result of arctan2 is renormalized so that the negative values are shifted. Now the Lidar starts at 0 and ends at 360 (Instead of 0 ->180->180->0)
   3. Subtract the angle of trigger/front, which is 180
2. The time of the point before/after the trigger is calculated
   1. Timestamp at start plus the duration of one cycle multiplied by the angle in xy-plane divided by 360 (t\_start + t\_cycle \* angle of point/360)
   2. The timestamps get renormalized to the trigger (i.e. trigger time -> 0)
3. The point cloud is mapped into the IMU coordinate frame (using the inverse of imu2velo)
4. Fix distortion
   1. We can calculate translation of each point by multiplying the velocity of each point by the timestamp relative to trigger (or time to trigger).
   2. We can calculate the angle by multiplying the angular rate in z direction by the time to trigger and form a rotation matrix.
   3. We combine the rotation and the translation into a matrix transformation on homogeneous coordinates.
   4. We transform all the points
5. Project the points back to lidar coordinate frame (with imu2velo)
6. In order to visualize the results, the Lidar points were transformed from the Lidar frame into the camera frame
   1. Transform the velodyne point cloud into homogeneous coordinates
   2. Use the t\_cam2\_velo matrix to transform to camera frame
   3. Remove all points with z (forward) smaller 0 to be left only with points in camera frame
   4. Project from the 3d homogeneous camera plane onto the 2d homogeneous image plane using P\_rect\_20 matrix and normalize the result over the homogeneous coordinate
   5. Transform into 3d, by dropping 4th dimension
7. Distances are calculated using the euclidean norm on the point cloud and coloured using the depth\_color function



## Problem 5

1. This has to do with two reasons. For one the intensity of the electric field on the lasers drops by the intensity squared, so the individual beams actually become more intense as you come closer to the scanner. Furthermore, the density of the point cloud increases, meaning there are actually more potentially harmful lasers hitting you.
2. The change of reflectivity caused by water can cause phenomena that are undesirable, such as specular reflections. Furthermore, if the laser actually penetrates the water and manages to return to the laser, it will suffer some accuray in the time of flight as water has a different reflective index, which means the light will travel slower through water.
3. The lidar may have an angle on close objects that the camera doesn’t have and vice versa. This can make it difficult to fuse the data. This will get worse, the further away the camera is from the lidar.