Deep Learning for Autonomous Driving - Project 1 Team 36

Mesut Ceylan
Legi: 18-748-012
ceylanm@student.ethz.ch

Adrian Hoffmann
Legi: 15-933-930
adriahof@student.ethz.ch

March 27, 2020

1 Task 1

Our code first loads the Velodyne data, separates the intensities from the 3D points, and augments the points to get homogeneous coordinates. Afterwards we multiply each point with the following matrix:

$$\begin{pmatrix} 0 & -1 & 0 & -\text{min y coordinate value} \\ -1 & 0 & 0 & \text{max x coordinate value} \end{pmatrix}$$

The effect of this two fold. We project the Velodyne data onto the x-y-plane and move the origin to the top left corner of the bounding rectangle of the points (in bird's eye view where the direction of travel points to the top of the bounding rectangle). The x axis goes from left to right and the y axis from front to back (both from the driver's point of view). We then put a grid over the whole bounding rectangle such that one pixel (= one field of the grid) equals a square with side lengths of 0.2 meters. Each pixel gets the intensity equal to the maximal intensity among all points contained in the pixel. You find the resulting figure here 1. Note that, since we have not received a height of the 'bird', we were not able to take perspective into account.

2 Task 2

2.1 Task 2a

We perform the following steps in our code:

1. Load the Velodyne data and port the 3D points into a homogeneous coordinate system.
2. remove all points that are behind the LiDAR (that is, remove all points with a negative x-value)
3. get each point onto the image plane of camera 2 by first multiplying each one with the matrix (*P_rect_20* * *T_cam0_velo*) and then dividing the result by its z-value
4. filter the projected points that don't fall onto the provided picture
5. use the provided labels and colors to add colored representations of the remaining points to the image

You can find the result here 2.
Note: For easier recognition we not only color the pixel a point falls onto but also the pixel to its right, bottom, and bottom-right.

2.2 Task 2b

First of all, we discarded the last item in the $objects$ list, since its category was *'DontCare'*. But we could easily add it to our procedure.
For each object we first define its preliminary bounding box with 8 points, all with homogeneous coordinates. Most

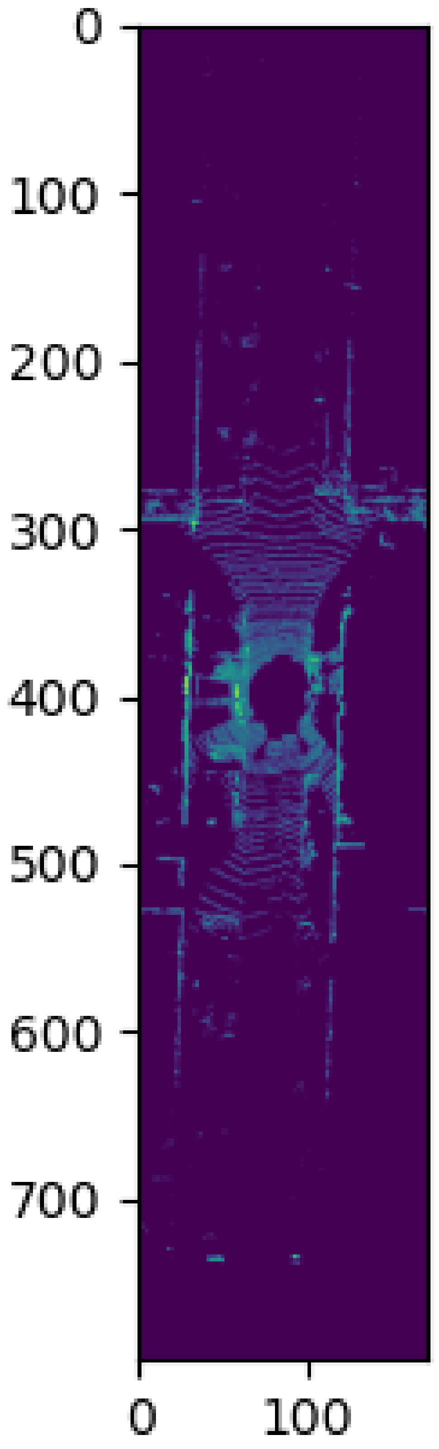

Figure 1: Bird eye view of the Velodyne data in task 1

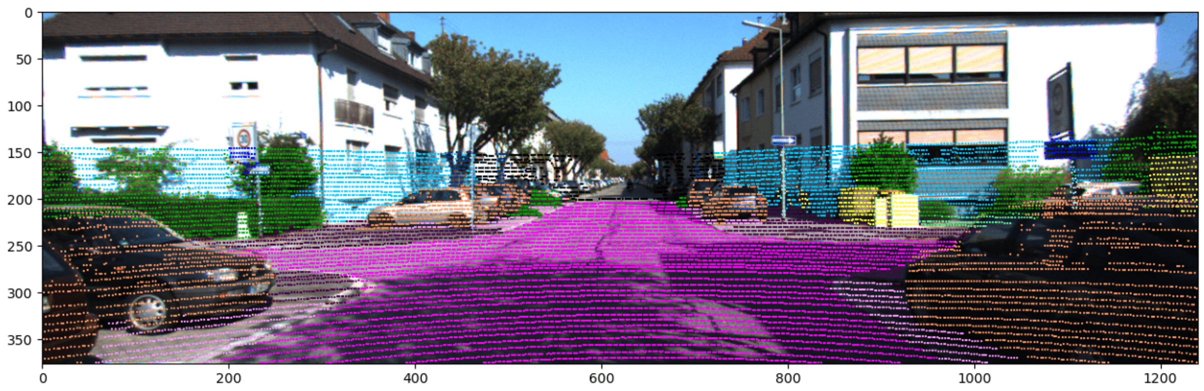

Figure 2: Image from task 2a with additional pixels colored for easier perception.

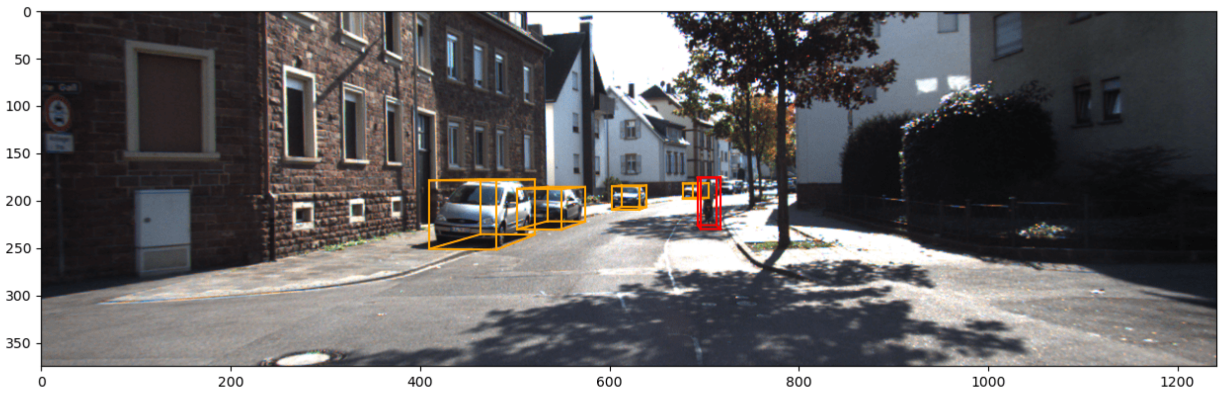

Figure 3: Image from task 2b. Orange is for the class 'Car' and red for 'Cyclist'

importantly, the center of the bottom rectangle of each preliminary box coincides with the origin of the coordingate system. We then utilize the following family of matrices to rotate and translate the preliminary boxes to their final positions:

$$\begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) & t_1 \\ 0 & 1 & 0 & t_2 \\ \sin(\theta) & 0 & \cos(\theta) & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The t-vector is the offset and θ the negative ry-angle of a bounding box.
As the second to final step, we used the *P_rect_20* matrix (along with dividing by a point's z value) to get all bounding boxes onto the image plane of camera 2. Finally, all that was left is to draw the boxes onto the image itself. You can find the result here 3.

3 Task 3

For this task we map each point onto the tuple (ϕ, θ), where ϕ is the angle spanned between the x-z-plane and the vector to the point and θ is the analogous to the x-y-plane. We first remove all points that are not in a cone in front of the car. Specifically this means that we a) remove all points with a negative x value and b) remove all points for which $|\phi| \geq 40°$. The $40°$ was found by trial and error so that the cone is narrow but no point that will fall on image_2 is removed. The motivation for filtering by ϕ is to remove noise for the next step which is k-means clustering with 64 centers (since we have 64 lasers) on the θ values of the remaining points. The centers are then sorted and their indices define the laser-IDs. To find the laser-ID of an individual point we simply search for the center that has the smallest distance to the point's θ value.

The remainder of the task is analogous to Task 2 except that we color even more pixels. For each pixel that coincides with a LiDAR point, we color that pixel and all its eight neighbours. The result can be seen here 3.

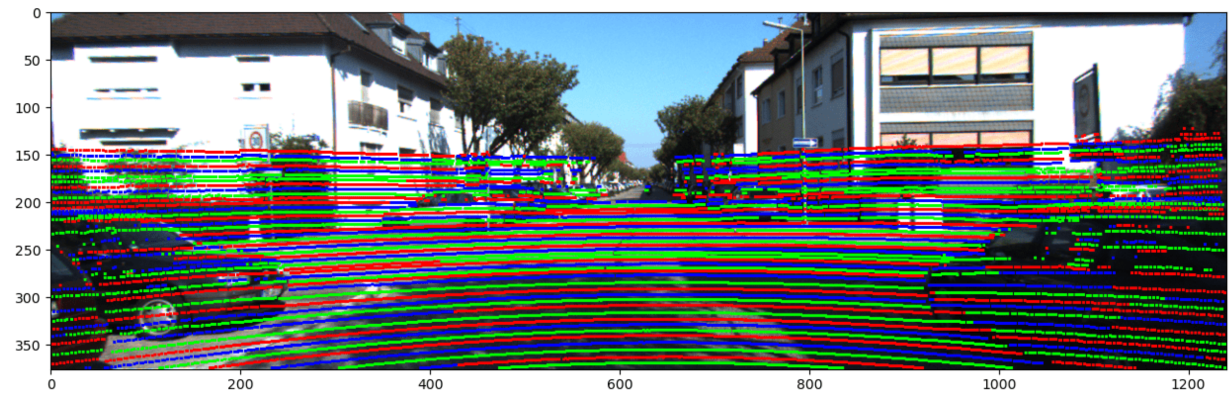

Image for task 3 with extra pixels colored

4 Task 4

Essential challenge of task 4 is to remove distortion of the 3D point cloud that is obtained Velodyne LIDAR. To tackle motion distortion caused by planar movement of the vehicle, we leveraged GPS/IMU measurements, particularly angular velocity around z-axis for rotation and velocity at x and y axis for translation martices that we built and via this matrices, we undistorted the points.

4.1 Projecting Distorted Points

As part of the Task 4, at first we projected distorted 3D point cloud to image plane and map the points into color coding according to their distance. To accomplish the result, we also filtered out the points that are lying out of the image plane of the specific frame.

During this calculation step, we used P_d as distorted 3D point cloud, we obtained rotation and translation matrix from "calib-velo2-cam" function then stacked them to have 4x4 matrix, and finally Camera 2 intrinsic parameters from "calib-cam-to-cam" function. After multiplication of matrices, we then normalized the obtained points along z-axis to obtain normalized points to project. Beforing conducting projection, we filtered out the points outside of the image plane. Finally, to map colors to the projected points w.r.t their distance, we utilized the "depth-color" function.

Basically, we projected distorted 3D point cloud to image plane with following equation:

$$\begin{bmatrix}u\\v\\1\end{bmatrix} = \frac{1}{z^c}\begin{bmatrix}\alpha&0&u_0&0\\0&\beta&v_0&0\\0&0&1&0\end{bmatrix}\begin{bmatrix}x^c\\y^c\\z^c\\1\end{bmatrix} = \frac{1}{z^c}\underbrace{\begin{bmatrix}\alpha&0&u_0&0\\0&\beta&v_0&0\\0&0&1&0\end{bmatrix}}_{\text{Intrinsic parameters}}\overbrace{\begin{bmatrix}\mathbf{R}&\mathbf{t}\\\mathbf{0}^T&1\end{bmatrix}}^{\text{extrinsic parameters}}\begin{bmatrix}x^w\\y^w\\z^w\\1\end{bmatrix}$$

Figure 4: Projection Equation-*source:lecture slides*

4.2 LIDAR Start Angle

As information provided in the project pdf file, we are in need of finding where LIDAR start scanning. We know that for each frame LIDAR starts scanning and when LIDAR looks exactly forward as camera looks, then camera triggered and afterwards LIDAR finished its scanning process. To understand in which angle LIDAR starts its scanning w.r.t position of camera, we first computed duration of LIDAR scanning. Then we computed duration difference between camera trigger time and LIDAR start time. Then, we multiply this duration difference with equivalent of one full turn of LIDAR (basically radian value of 360 degrees which is 6.283), and divided it with full LIDAR scanning time. In this approach, we obtained angle of the LIDAR in radians. Finally, we convert radian values to degrees by multiplying radian value with 180 and divided by pi.

To explain this procedure, we followed this notation: t_{start} for LIDAR starting time, $t_{lidarend}$ for LIDAR ending time, $t_{camtriggertime}$ for the time that camera is triggered. We computed the angle of points as following:

$$t_{lidarfulltime} = t_{lidarend} - t_{lidarstart} \tag{1}$$

$$t_{diff\_radian} = (t_{lidarstart} - t_{camtriggertime}) * \frac{6.283}{t_{lidarfulltime}} \tag{2}$$

$$t_{diff\_degree} = t_{diff\_radian} \times \frac{180°}{\pi} \tag{3}$$

Finally, after all these computation we observed that each time LIDAR starts its scanning from $-180°$. This finding means that LIDAR starts each scanning completely from opposite direction of camera front view.

4.3 Point Order

According to project pdf file, we know that order of the 3D points is lost. To tackle this challenge, we benefit from findings from LIDAR Start Angle section. We found that LIDAR starts scanning at $-180°$ and finishes at $180°$. As information provided, LIDAR rotates clockwise direction, thus 3D point cloud that has been built can be sorted according to increasing order of the angles of points. We are aware that motion of the vehicle is planar, hence we only need to analyze the points in x-y plane in order to find respective angle.

To find the angle of each 3D point, we first extracted x coordinates of points, x_i and y coordinates of the point, y_i. We computed the angle of the point as following:

$$\tan(\theta_i) = \frac{y_i}{x_i} \tag{4}$$

$$\tan \text{inverse}(\tan(\theta_i)) = \tan \text{inverse}(\frac{y_i}{x_i}) \tag{5}$$

$$\theta_i = \tan \text{inverse}(\frac{y_i}{x_i}) \rightarrow \theta_i = \arctan(\frac{y_i}{x_i}) \tag{6}$$

4.4 Undistorting Point Cloud

As it is indicated, 3D points are distorted due to motion of the car. Speed of motion and scanning rate are the factors impacting intensity of the distortion. In this section, we introduce how we computed translation and rotation matrix by using high precision measurement provided by GPS/IMU.

First of all, we know that vehicle moves planar, thus we are going to ignore rotation along the z-axis and solely focus on rotation on x and y axis.We defined rotation formula into matrix form as rotation matrix R_i and rotation angle α_i per point and angular rate around z-axis ω_z(rad/s) as following:

$$R_i = \begin{bmatrix} \cos(\alpha_i) & -\sin(\alpha_i) \\ \sin(\alpha_i) & \cos(\alpha_i) \end{bmatrix} \tag{7}$$

$$\alpha_i = \frac{\omega_z \times \theta_i}{2\pi} \tag{8}$$

Final rotation matrix has 4 elements per point. We know that rotation only happens on z-axis (the yaw) such that we computed rotation angle α_i by multiplying it with angle of the point divided by 2π. In addition, since LIDAR moves clockwise, we took negative of the angle of points when computing related parameters. Secondly, we computed translation matrix T_i that consists of translation on x-axis t_{i_x} and translation on y-axis t_{i_y}.

$$T_i = [t_{i\_x} \quad t_{i\_y}] \tag{9}$$

To compute translation values, we obtained forward velocity v_f and leftward velocity v_l. Then, we multiply them with angle of the point and divided them by 2π. We computed translation matrix T_i elements as following:

$$t_{i_x} = \frac{v_f \times \theta_i}{2\pi} \quad t_{i_y} = \frac{v_l \times \theta_i}{2\pi} \tag{10}$$

Final shape of the translation matrix is 2x1. As we obtained rotation and translation matrix, we formed them together to be able to multiply this matrix with distorted points. Merged matrix has the shape of 2x3.

$$[R|T]_i = [R_i T_i]$$

Finally, we are required to undistort all the distorted 3D points into undistored ones. For this step, we multiplied distorted points with our rotation and translation matrix.

$$P_{i_{undistorted}} = [RT]_i \times P_{i_{distorted}}$$

When we obtained undistorted points $P_{i_{undistorted}}$, we need to compute projection equation with undistorted points. To project undistorted points, we computed projection equation as following:

$$P_{projection} = K_{cam2} \times [R|T]_{velo\_cam} \times P_{undistorted}$$

In above formula K_{cam2} is camera 2 intrinsic parameters obtained from *"calib_cam_to_cam"* function, $[R|T]_{velo\_cam}$ rotation and translation matrix from *"calib_velo_to_cam"* function, and undistorted points $P_{undistorted}$ from above-stated procedure. Finally, we obtained $P_{projection}$ points and projected them along with color coding according to their distance.

4.5 Results

As a result of undistortion process that explained in previous subsection, we obtained 3 different output image by testing stated frames, namely frame 37, 310 and 320. Below, you may find three different image set per frame. In each set, first image is raw image that is provided, second image has distorted 3D point cloud projected into image frame, lastly third image has undistorted 3D point cloud projected into image frame.

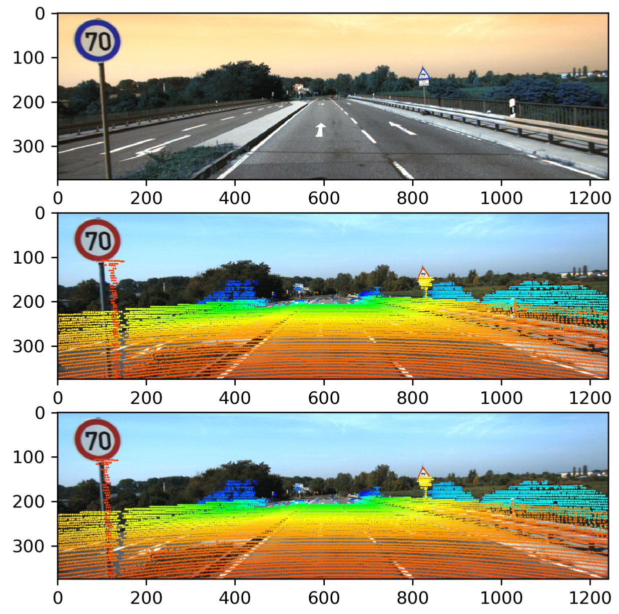

Image Set of Frame 37

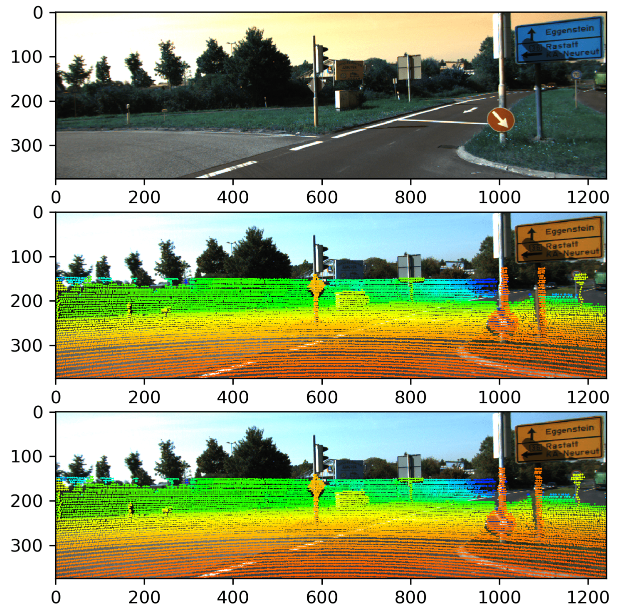

Image Set of Frame 310

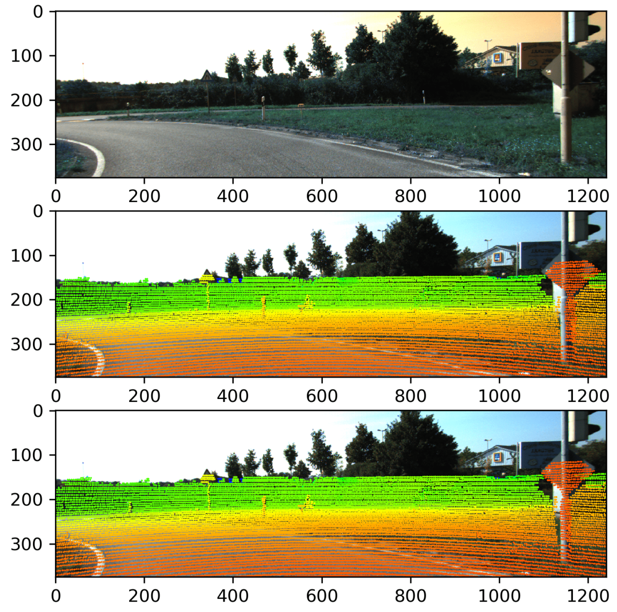

Image Set of Frame 320

5 Task 5 (Bonus)

1. When the laser beam travels through the air it gets weaker and weaker due to small particles in the air which scatter parts of it. Therefore the laser doesn't have the same intensity for a pedestrian compared to someone who stares into the LiDAR from a very short distance.
2. Wet roads tend to be reflective. For a camera, this means that you can see the reflection of an object, that shouldn't be there, on the street. In an extreme case this could lead the car to make a wrong decision. Also, the exposure could become a problem: if the sun reflects on the street a camera has to decide whether it wants to let so little light in that it still can see details of the road (mainly lines) but then the surroundings are dark (meaning that e.g. signs aren't seen). Or it could let in a lot of light which gives us a clear view of the surroundings but then leaves the road as completely white surface.
   For a LiDAR the wet road could become almost a mirror. This means that laser beams that hit the road will mostly be reflected away from the sensor similar to if it had hit a mirror. As a consequence, it might be that too little of the beam will be reflected back to the LiDAR meaning the measurement is maybe too weak to register. Also, in some cases, a beam might bounce a few times and then find the way back to the LiDAR where it will be noise for another measurement.
3. Since they have different points of view, some objects might be in a blind spot for one sensor but not the other and vice versa. Hence the projection of the measurements of one sensor might become non-sense when projected onto the measurements of another sensor. This is problematic when you e.g. base decisions on the premise that both sensors see a hazard. This problem becomes more severe when you increase the space between the senors because because points of view get more and more different.