Project 1: Understanding Multimodal Driving Data

Berkay Berabi, Tommaso Macrì
Deep Learning for Autonomous Driving
ETH Zurich

March 27, 2020

Problem 1. Bird's Eye View

For this task, we firstly decided to ignore the z-coordinates of the points, since it is a bird eye view and the height of the points do not matter. However, it is still possible that two different points in the lidar point cloud, can be on top of each other(e.g x and y coordinates are the same and z is different.) In such a case, the conflict is resolved in favor of the higher point, since the highest point will hinder one from seeing other points when looking from above. Secondly, we define a box around the lidar and filter out the points that are not in the desired box which is predefined by the front and side ranges in the code. At this point, we transform the points from lidar coordinates to BEV image coordinates because the coordinate system of the image and the lidar is different. Lastly, we compute to which pixels each point belongs to, by dividing the coordinates by the chosen resolution. Finally, we go over each point, and assign its pixel the reflectance value of that point in the lidar data and plot the bird eye view image in gray color map. All these procedures can be found in the code, where we defined a function "lidar2bev" with inputs: lidar point cloud, resolution used to plot the BEV map, and the front and side ranges used for plotting. The resulting image can be found in Figure 1.

Figure 1: Resulting BEV image.

Problem 2. Projection onto Image Plane

a. Semantic Segmentation: Displaying Semantic Labels

In this problem, we project the point cloud given by the lidar "segmentation data" onto the the given image. The objective is to plot the points with different colors, according to semantic information of the scene in the figure.
For this, we extract all the needed information from the given Python dictionary:

- The point cloud,
- The Image
- The intrinsic projection matrix to Cam 2 after rectification, (3x4) numpy.array object
- The homogeneous velodyne to rectified camera 0 coordinate transformation, (4x4) numpy.array object
- The semantic label of each point within the scene, (num points,) numpy.array object
- The dictionary mapping numeric semantic labels : BGR (reversed RGB) color for visualization

- The dictionary mapping numeric semantic labels : string class.

At the beginning, we project the 3d homogeneous points in lidar coordinate system to 3d homogeneous points in camera 0 coordinate system by multiplying the points with the matrix T_cam0_velo and then, we project the points to Cam2 coordinate system by multiplying with P_rect_20. We then get rid of all points that are behind the camera retaining only points, whose x-coordinates are higher than 0.27 because this is the distance between the camera2 and the lidar. With the computer vision library we plot the points as circles using the semantic label to determine the circle's color. Finally we plot the image and the results are very good and shown in Figure 2 and 3.

Figure 2: Image 2.

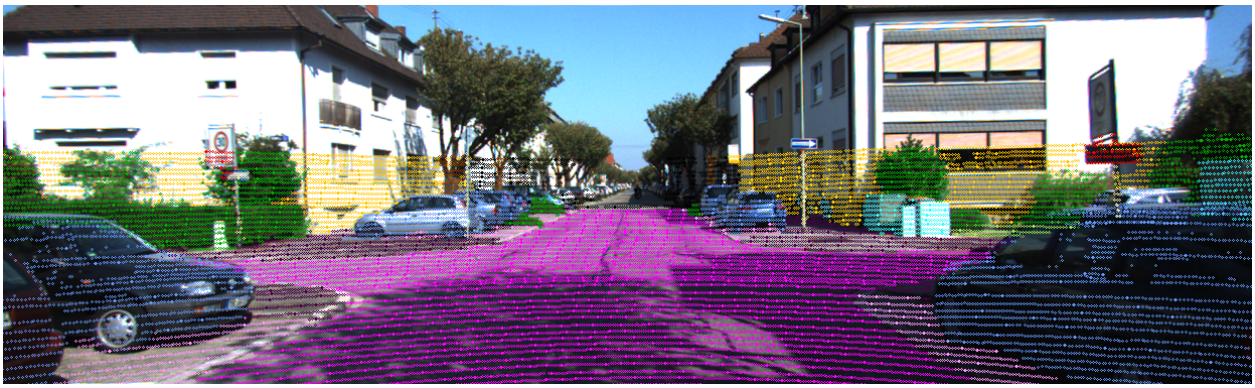

Figure 3: Semantic segmentation.

b. Object Detection: Drawing 3D Bounding Boxes

In this problem, instead, we use the detection data to project 3d bounding boxes onto an image that detect the given vehicles, pedestrian and cyclists within the scene. For this, first of all we ignore the object classified as "Misc" or "DontCare" as they are not supposed to be shown. We then extract the height, width, length, rotation around the y axis in cam coordinates, the center of the 3d object in camera coordinates. With this information, we compute the corners of the bounding boxes (4 for the bottom face and 4 for the upper face)

relative to the center. As next step, we rotate the corners by the given amount of rotation around the y-axis and add the center to all corners in order to obtain coordinates relative to the cam2. With the cv2 function "rectangle" and "line", we plot the bounding boxes which are shown in Figure 4.

Figure 4: Bounding boxes.

Problem 3. ID Laser ID

For this exercise, we consider the lidar again and its different 64 channels. We want to identify by which specific channel each point is captured. One can obtain this information by considering the angle between the lines from lidar origin to the point and the projection of this line to the xy-plane.
With trigonometric equations, we find these angles for all points and knowing the vertical angular resolution of the lidar from the specifications given, we find the points' channel id by subtracting the angles with the minimum scanned angle and then dividing by the angular resolution. Later we plot the laser ids by alternating colors and the result can be seen in Figure 5

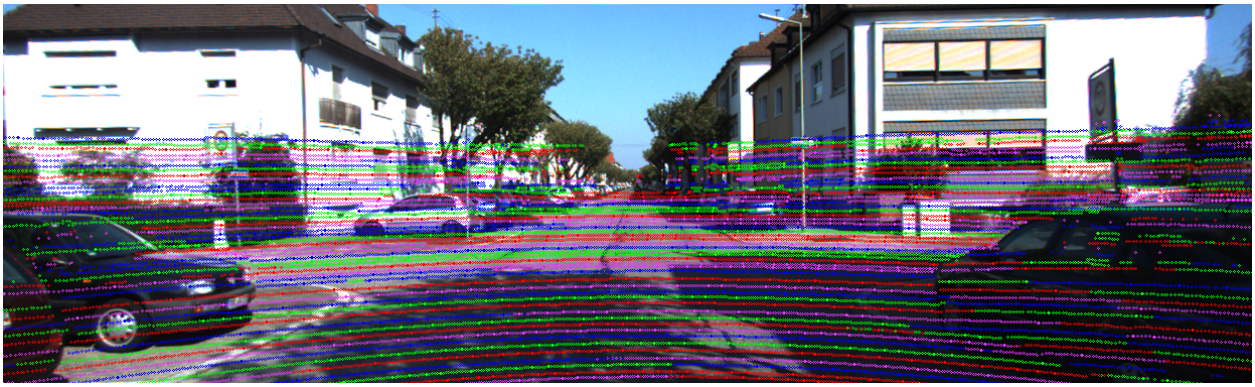

Figure 5: Laser IDs.

Problem 4. Remove Motion Distortion

In this problem, we consider the delay between the time when each image is taken by the camera and the timestamps when the lidar points are acquired. Projecting the laser point cloud onto the image without considering this delay, would cause a distortion and we aim to correct this. The points detected by lidar are captured in a 360 sweep where the car is possibly rotating and translating. This means that there might be points that are captured before as well as after the camera was triggered. Basically, for each point we need to find at which timestamp relative to the timestamp at which the camera was triggered, it was captured and then one can transform the point to the timestamp, at which the camera was triggered. In order to know how much we need to translate and rotate the points, we assume that the car is moving at constant velocity and rotating at constant angular rate, too. These are measured by GPS/IMU sensor and are available in the given data. If we know the relative timestamp we can basically multiply this timestamp with velocity and angular velocity to obtain the required corrections for a particular point. The trigger angle can be calculated from the given timestamps as

$$triggerangle = (t_{lidartrigger} - t_{lidarstart})/(t_{lidarend} - t_{lidarstart}) * 360 \tag{1}$$

After computing the trigger angle, we can shift the angles for each point, such that the trigger angle becomes 0 degree and now the timestamp for a point is

$$t_{point} = t_{lidarstart} + (t_{lidarend} - t_{lidarstart}) * (\alpha/(2 * \pi)) \tag{2}$$

where α is the angle relative to the trigger angle.
Finally, we calculate the time difference between the timestamp where the camera was triggered and a point was taken. We multiply the velocity and the angular rate with the time difference and obtain the required translations and rotations. Later we create the transformation matrix for each point and we correct the points and plot the results, also visible in Figures 6 and 7, 8, 9, 10, 11.

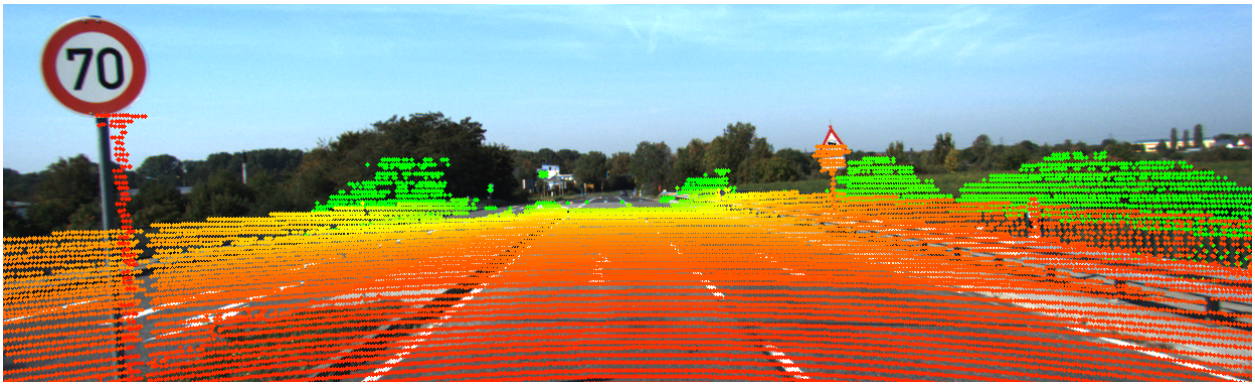

Figure 6: image 0000000037 distorted.

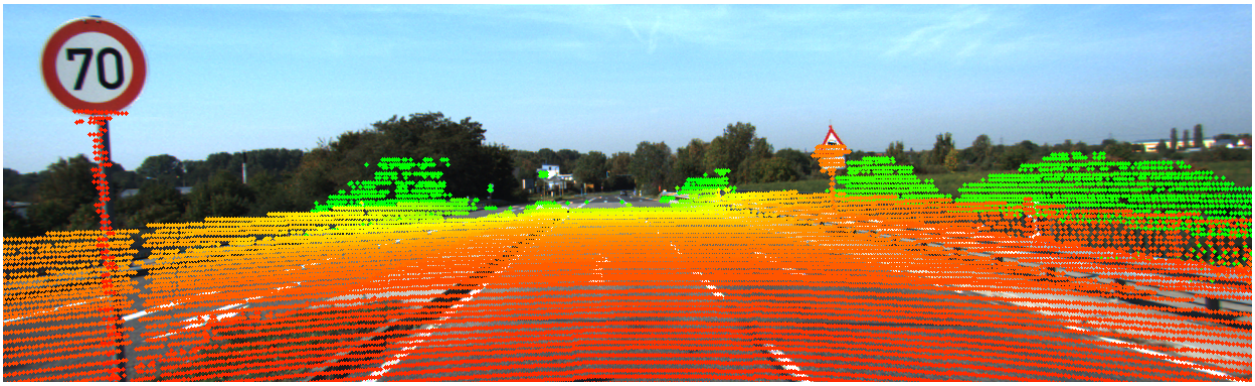

Figure 7: image 0000000037 corrected.

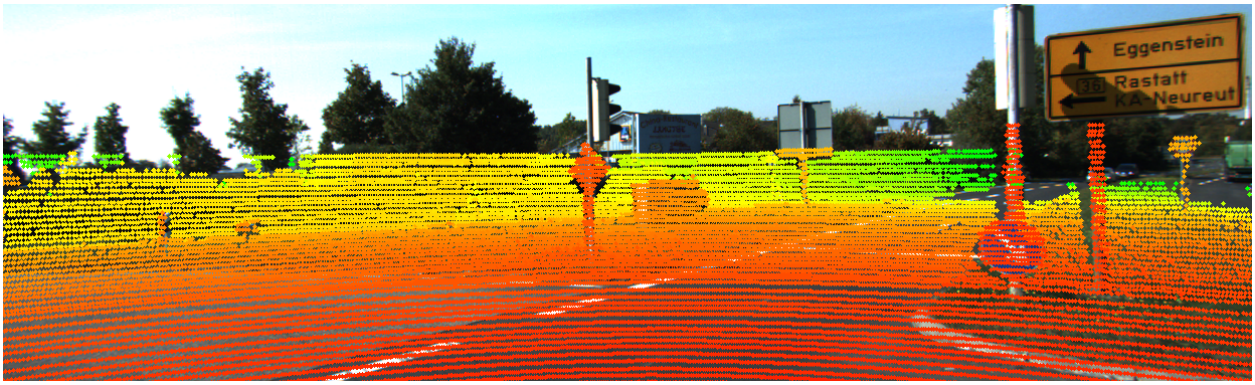

Figure 8: image 0000000310 distorted.

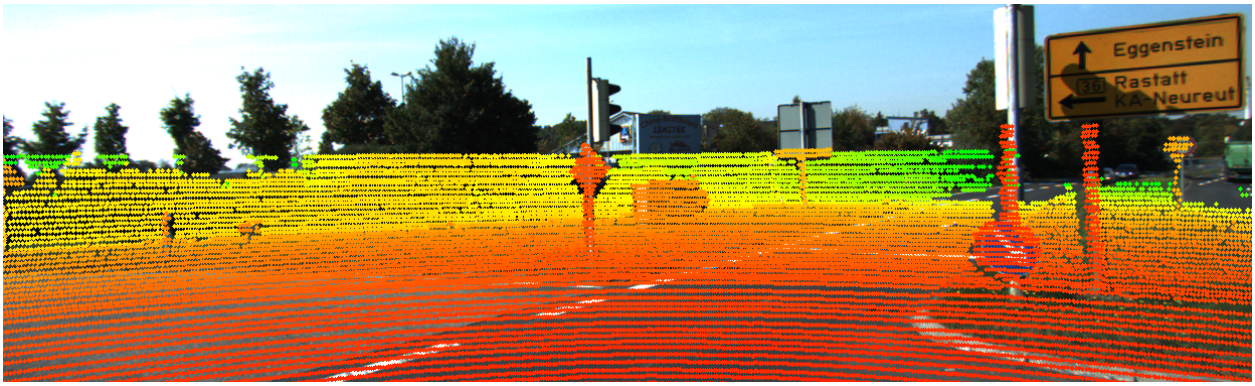

Figure 9: image 0000000310 corrected.

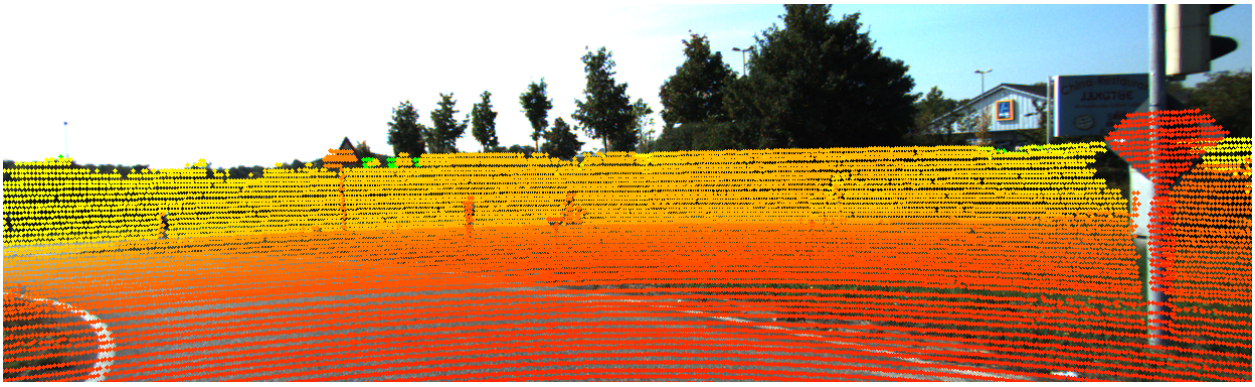

Figure 10: image 0000000320 distorted.

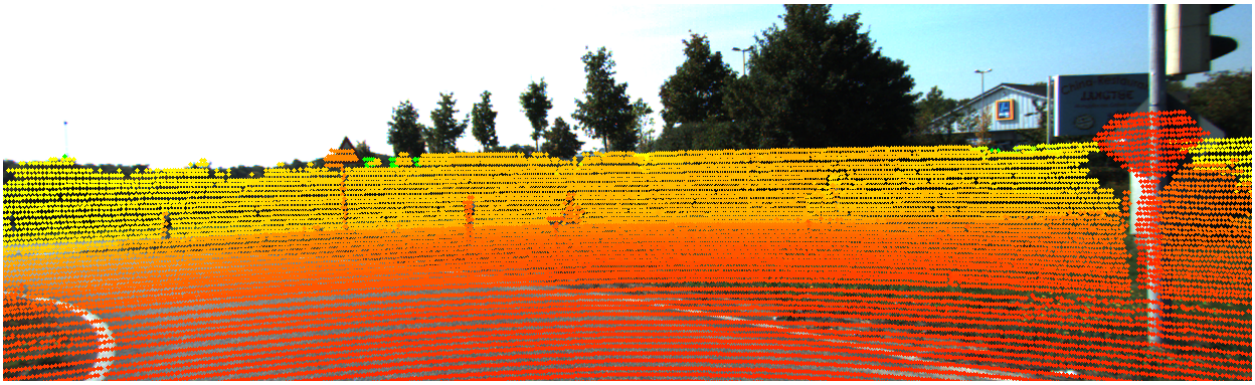

Figure 11: image 0000000320 corrected.

Problem 5. Bonus Questions: you can earn 1 more point

1. Eye safety: it is not safe for the human eye to look at a spinning LiDAR when it is too close. Why the risk is higher when we are closer to the sensor?

The lidar emits high energy light rays (laser) in order to detect objects in the surrounding. The rays are high energy for performance and reliability reasons but for this reason they can damage the eyes. When the lidar is closer to the eyes, it is more dangerous because the rays are thinner and the energy of a light decays with the distance it travels and therefore the specific energy (normalized by the ray circular area) is higher.

2. Wet road poses challenges for both cameras and LiDAR, what are the challenges and why?

Water drops on wet roads can reflect other light sources, as sun or cars' or environment lights. This can cause confusion for sensor that rely on visual information as lidars or

cameras. Besides this, the drops might also reflect lasers emitted from lidar. In addition to this, wet roads have less texture, which also lowers the camera performance.

3. In this exercise, you have projected LiDAR points onto images. In the setup in Fig.1, the LiDAR sensor and the Cameras are non-cocentered – it can never be exactly non-cocentered. What problem this may cause for the data projection between the two sensors (LiDAR and Cam2 for instance)? Do you think this problem will be more severe or less severe when the two sensors are more distant from each other?

Having the camera and the lidar not in the same point (co centered) means that projecting the lidar points onto the camera requests a transformation matrix multiplication. this can be a computational problem as well as it can distort the points if the matrix has errors (errors that come from the calibration or manual measurements of the distance between the camera and the lidar). At this point, the more the two sensors are far each other, the greater the error would be.