PROJECT 1

Understanding Multimodal Driving Data

Deep Learning for Autonomous Driving

Marian Kannwischer

Fabian Lindlbauer

25.03.2021

This project report entails the explanations of the programmed software to solve the provided tasks.

Table of contents

Sensor set-up for the following problems

The following image illustrates the sensor configuration used for the problems discussed in the following report. All references to coordinate systems refer to the data displayed in this figure.

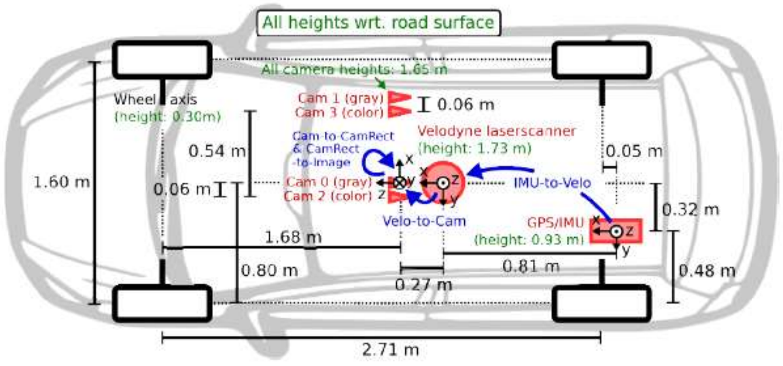


FIG 1. Sensor set-up of the autonomous car. Image obtained from the problem description of problem set 1.

Code description

project1.py

Script to run all the functions necessary to obtain the results for project one. The results are displayed on the screen and saved into the directory where the files are stored.

objects.py

Entails the model and its functions that extract the data from the provided data dictionary and computes/illustrates the results of the given tasks. The model functions are implemented such that a mapping between different coordinate systems can be provided via the model function arguments. For any detailed information on the model function, please refer to the specific function headers as they provide all the necessary information to understand what to expect from each function.

3dvis.py

Modified file given for this task. This file creates a visualization of the lidar points in 3D space with points coloured according to their semantic labels and further displays the bounding boxes of all detected cars in the scene.

task_4_script.py

The solution to this task is implemented in task_4_script.py. One can set the frame either by adjusting the constant parameter at the top of the file or by calling the script from the command line with “python task_4_script.py [frame]”. Furthermore, one can configure whether the Velodyne forward timestamp or the image timestamp should be used.

Problem 1. Bird’s eye view

The task of problem 1 was to create a bird’s eye view of the provided lidar point cloud data. The resolution of the plot is 0.2 m in both image directions, i.e. x and y coordinates according to the lidar coordinate systems. If two or more lidar points lie in the same 0.2x0.2 m bin, the lidar point with the highest intensity is selected.

The strategy to achieve the image shown below was as follows. Firstly, the min and max x and y coordinates of the lidar point cloud were computed. The absolute difference in length between the min and max values was computed and then divided by the given resolution of 0.2 m. This computation yields the number of bins in both x and y direction. In the last step the lidar points were then assigned to their xy bin according to their x and y coordinates. Figure 1 below illustrates the result given as output of the code.

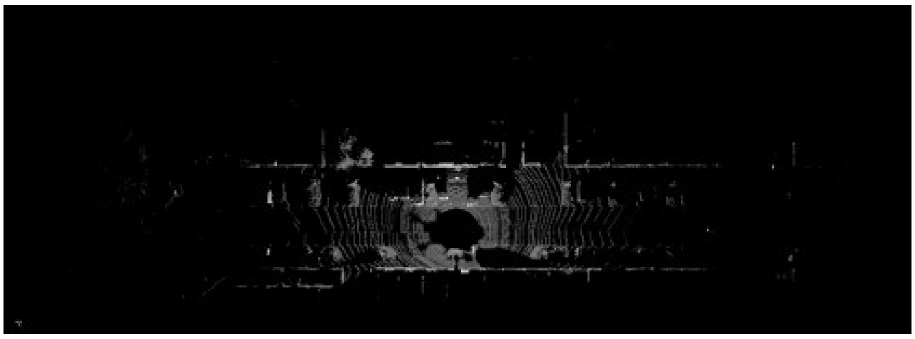


FIG 2. Illustration of the lidar point cloud in bird’s eye view. The resolution in both x and y direction with respect to the lidar coordinate system is 0.2 m.

Problem 2. Visualization

Task 1.

The goal of task 1 was to map the given lidar point cloud data onto the image produced by camera 2 and label each point according to its semantics in the image. E.g. the area in the image that corresponds to the street should feature violet points. The different colour mappings for different scene themes are provided in a Python dictionary.

In order to project only relevant points onto the image, i.e. points that are indeed seen by the camera, lidar points of a specific angle are extracted from the point cloud. For camera 2 the chosen angle was 40 degrees. This angle was obtained by testing different camera view angles as no camera data was available in the problem description sheet. Hence, points that lie behind the camera must be omitted as they would be projected onto the camera image plane as well leading to a wrong result. Furthermore, the colour index of each point according to the *color_map* dictionary entry was extracted and stored in a separate variable. These entries were then used to colour the points. The extracted points were then projected onto the image plane of camera 2 in three steps. Firstly, a transformation into the camera 0 coordinate systems using *T_cam0_velo* was applied. Secondly, the obtained coordinates were then projected onto the camera image plane by multiplying the data with the matrix *P_rect_20*. The last step included the scaling of the x and y coordinates of the transformed lidar points by dividing them by the 3rd column entry, which exists due to the usage of homogenous coordinates.

The transformed points were then drawn onto the image using Python's Pillow library. As the colours in the provided data dictionary are given in the colour code format BGR, the colours were first transformed into the traditional RGB format. For this purpose the function *bgr_to_rgb_colors* was made use of. The results obtained are shown in FIG. 3. below.

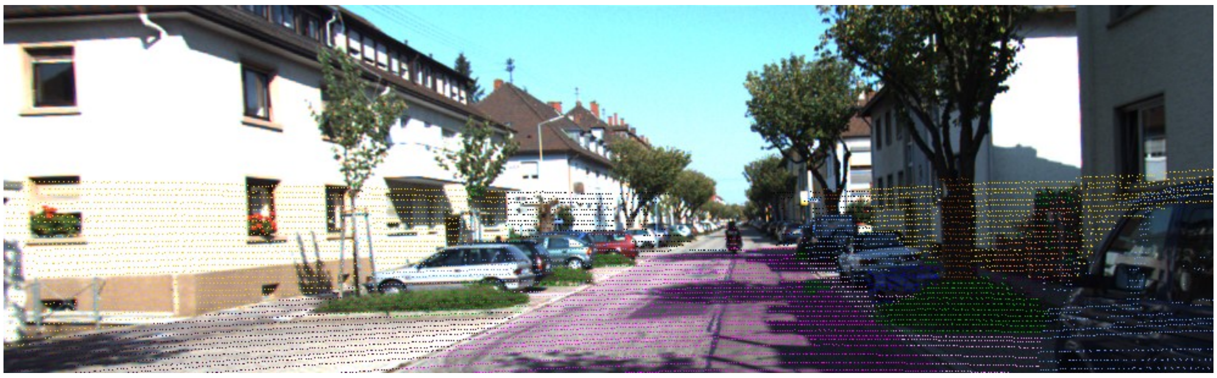


FIG 3. Image taken by camera 2 with projected data points recorded by the lidar sensor. The points are coloured according to their respective semantic label.

Task 2.

For this task 3D bounding boxes had to be placed in the image for all cars in the scene. For the projected lidar points the exact same code, which was also used for Task 1, was reused.

The starting point for drawing the bounding boxes onto the image were the x and z coordinates of each cars' bottom centre in the scene of camera 0 as well as the rotation angle of the car with respect to the y axis. In order to figure out the edge point coordinates of the bounding boxes, the coordinates were first computed according to a newly introduced coordinate system x-y, where x and y represent the x and z coordinates of the camera coordinate system respectively. The edge points could be computed by adding car width/2 and car height/2 to the edge points respecting positive and negative directions. Afterwards, a 2D rotation operation was performed according to the given rotation angle. The described operations yield the correct coordinates in 3D space of camera 0 for the car bottom. By adding the height of the car to these coordinates, the top bounding box coordinates could be obtained. By applying the matrix *P_rect_20,* the computed bounding box coordinates were projected onto the image plane of camera 2. The sketch below illustrates the chosen bounding box coordinate labelling. This choice was mainly driven by the input requirements of Pillow's polygon drawing function. Bounding boxes were drawn connecting the bottom 4 and top 4 corners first and then connecting these polygons together by drawing the vertical lines. The following sketch illustrates the choice of the bounding box coordinates and FIG 5 illustrates the results of the drawn bounding boxes.

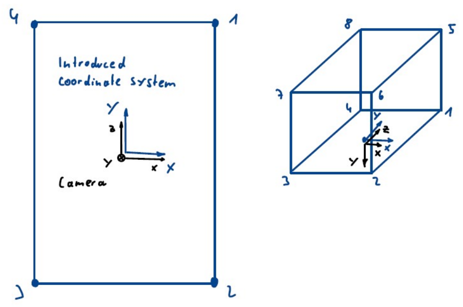


FIG 4. Bounding box counting strategy.

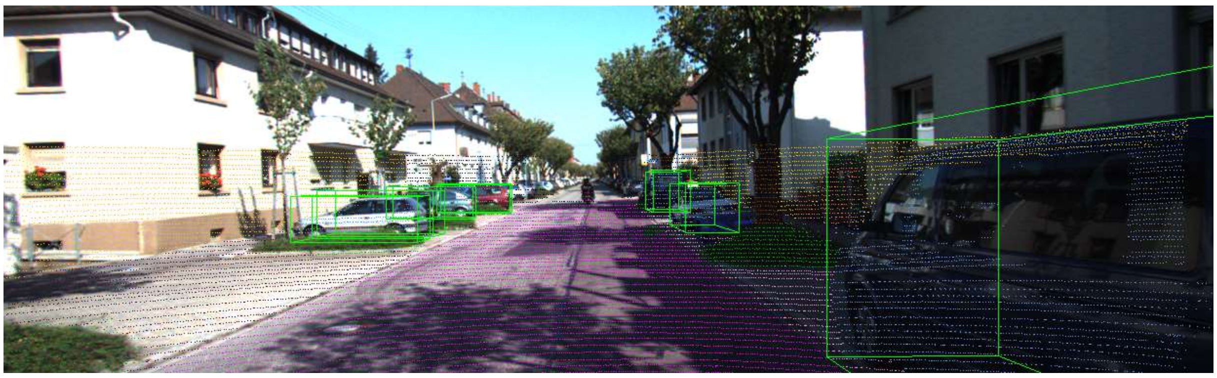


FIG 5. Image taken by camera 2 with projected data points recorded by the lidar sensor. The points are coloured according to their respective semantic label. Car objects are identified by green bounding boxes.

Task 3.

The requirement of task 3 was to complete the code given in file *3dvis.py* and visualize a 3D illustration of all the collected lidar points to obtain a better understanding of the whole scene.

The most important aspect to obtain the results shown below is the transformation from 3D camera coordinates to the coordinate system of the lidar sensor. To achieve this result, the matrix *T_cam0_velo* had to be inverted and then applied on the camera 0 coordinates.

(a) (b)

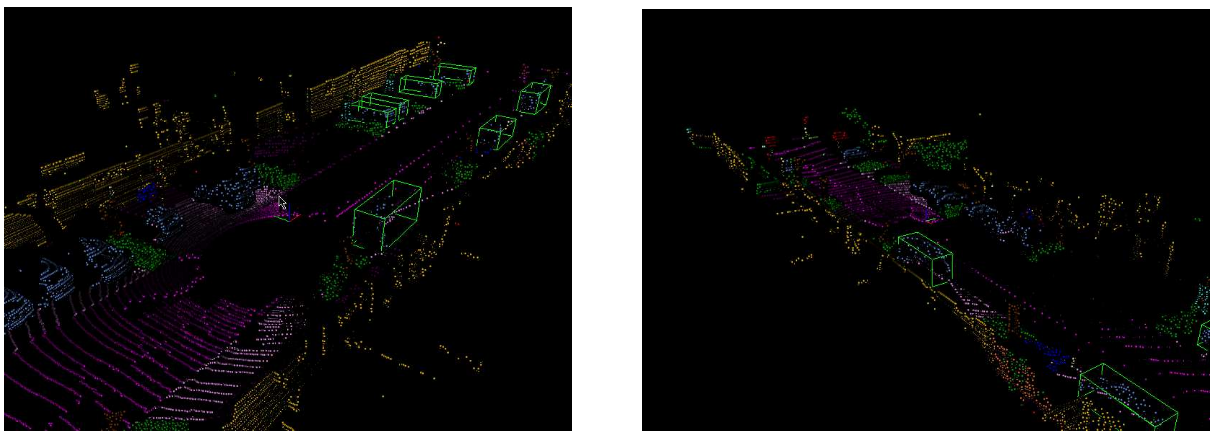


FIG 6. 3D visualization of the car environment using the modified code from file 3dvis.py. **(a)** Illustrates the forward looking direction of the car (positive camera 2 z direction) and **(b)** shows the backward looking direction (negative camera 2 z direction).

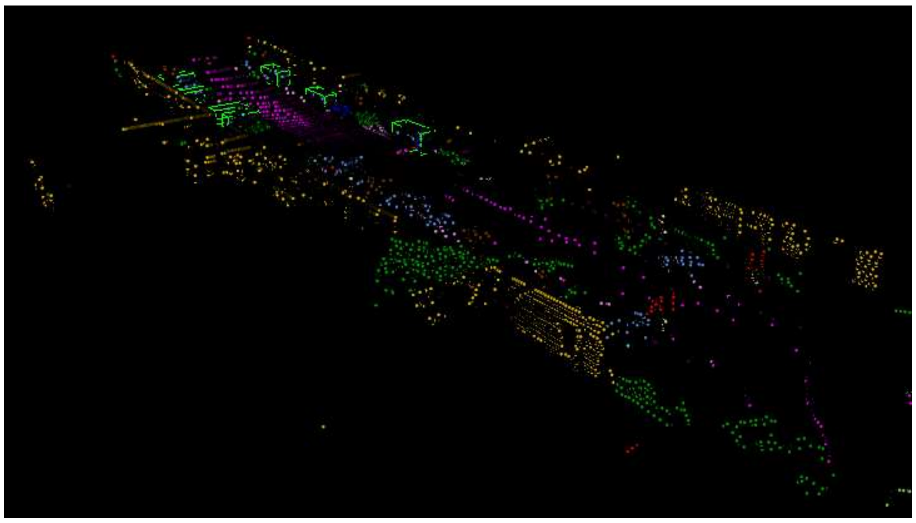


FIG 7. 3D visualization of the car environment using the modified code from file 3dvis.py. This image shows all the data points collected at the current position of the car.

Question from the task description: The 3D visualization clearly reveals that one car on the left hand side of the car was not detected by the neural network. The first 3 cars are detected, however, between the 3rd and 4th car one car was not recognized. This is a bit hard to see in the camera images, but the lidar point cloud allows seeing this car.

Problem 3. ID Laser ID

In problem 3 the task was to map the points of the lidar point cloud onto the image plane of camera 2 and colour them according to the laser ID of the lidar sensor they belong to. Laser IDs were assigned 4 different colours to clearly identify them (see figure below).

The strategy to solve this task was to exploit the sequence of lidar points stored in the provided Python dictionary. The first point recorded for a particular laser ID points into the positive x direction at an angle of 0 degrees (if this point exists), otherwise the first point can be found at a particular angle around the z axis (this is for instance the case for the highest lidar ID (top line)). Points are then assigned the same colour until a certain threshold change in the y direction with respect to the image plane is detected. This threshold change is looked for at a certain x interval in the image plane. If the threshold change is detected inside region (1) (see FIG 8 (a) below) then the drawing colour is changed between these two points (option 1, options are the black circled numbers in FIG 8). If no point can be found inside region (1) one of two different conditions can be met (options 2 and 3). One, if the considered point is to the left of region (1), i.e. in region (2), and the previous point is to the right of region (1), i.e. in region (3) then the colour must be changed, because a new line was started (option 2). Two, if the considered point is to the left of region (1), i.e. region (2), and the previous point is inside of region (1) then the colour is going to be changed in case no threshold change was found inside of interval (1) (option 3).

The threshold height in y direction with respect to the camera coordinate system is calculated by taking the height of the last 10 points into consideration. The height difference between the maximum and the minimum y coordinate is computed and then added to the standard deviation of the y coordinates of these 10 points. If the difference in y coordinates of the current point and the previous point is bigger than the computed value of the 10 previous points the drawing colour is changed. Due to the fact that this height checking is only done for the interval (1) it does not cause too much overhead on the overall algorithm.

In the following figure the above descriptions are represented in a graphical manner.

(a) (b)

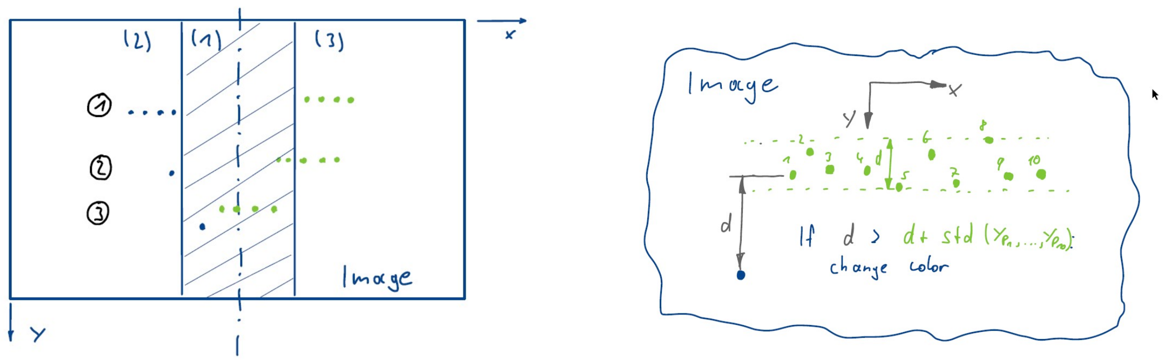
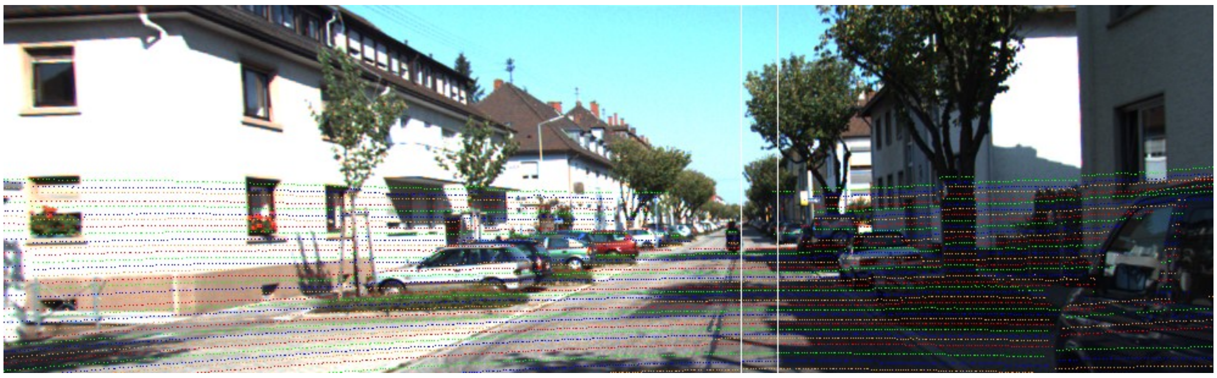


FIG 8. (a) Illustrates the 3 different options that are considered for a colour change when mapping the points from the lidar cloud into the camera coordinate system. **(b)** Illustration on how a decision on a colour change is made based on the y coordinates of the considered points.

(a) Lidar IDs with the vertical line interval to identify the colour mapping.



(b) Lidar IDs without interval indication.

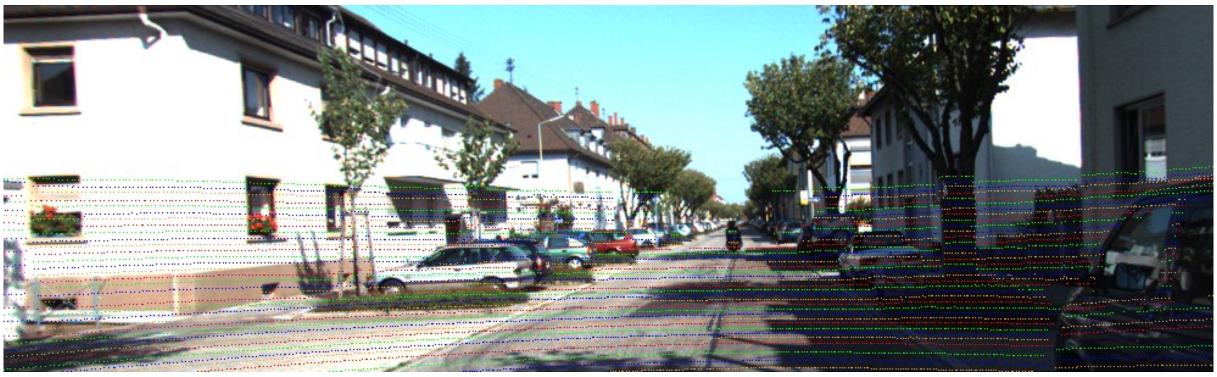


FIG 9. Recorded lidar points mapped onto an image taken by camera 2 and coloured according to the laser ID. Each colour identifies one laser angle. A sequence of green, blue, orange, and red is used to clearly identify the lasers IDs. This sequence is repeated for all the IDs.

Problem 4. Remove Motion Distortion

Task

Mapping a LiDAR point cloud onto an image has some implications. Conducting a full scan takes roughly 0.1 seconds for the employed Velodyne system. If we take for example a car travelling straight at 30 km/h it covers 0.83 m per rotation. So, if the Velodyne would start the scan facing forward and obtain one point right at the beginning and one at the end of the scan with the same recorded distance, there would be a 0.83 m difference in reality! Hence, the translation and rotation of the car is not negligible. When naïvely transforming only from the Velodyne coordinate frame to the camera 2 coordinate frame, this results in errors, for which an example can be seen in FIG 10.

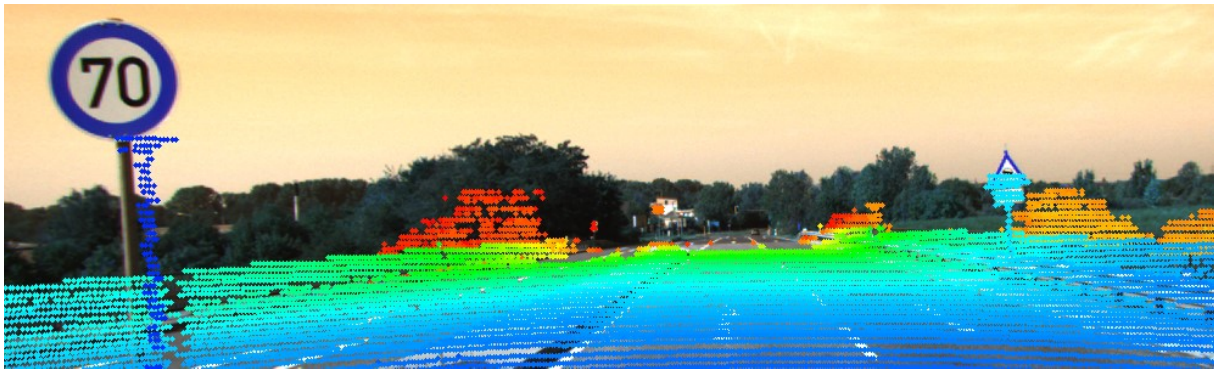


FIG 10. Naïve transformation without 'untwisting' results in errors, here visible for the speed sign.

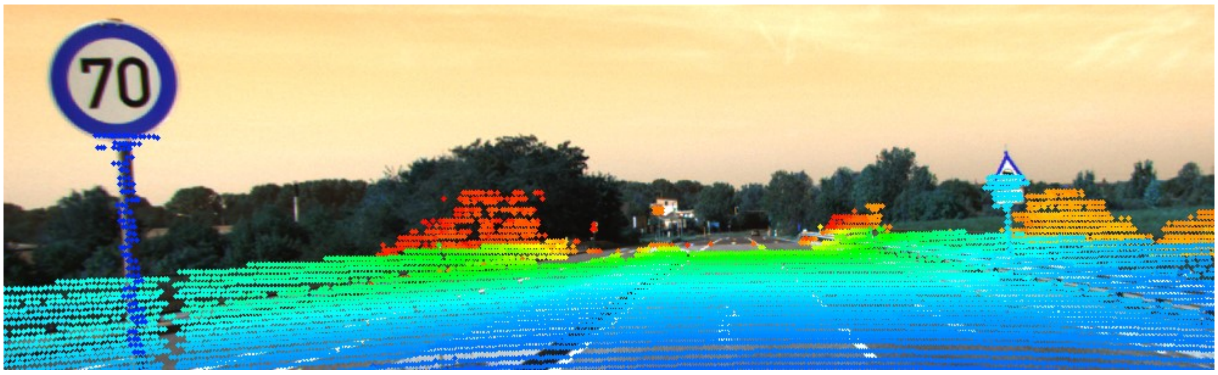


FIG 11. 'Untwisting' before projection.

Solution

First of all, it is necessary to know how the car is translating and rotating for a given frame. This data is provided by the combined GPS/IMU unit and is assumed constant for the relevant timeframe concerning each image. Theoretically, one has to transform from the oxts frame to the Velodyne frame (which we did in our code), but since the frames are well aligned, this does not affect the output much.

Next, one needs to figure out when a LiDAR point was registered relative to when the image was captured. Combined with the velocity $v \in \mathbb{R}^3$ and angular rates $\omega \in \mathbb{R}^3$ of the car, this time difference can be used to create a transformation that 'untwists' the point to the Velodyne coordinate frame at the time of image creation.

First, the duration of a whole rotation can be determined by subtracting the timestamp of start from the timestamp of the end of the scan. For frame 37, this results in 0.1035 seconds which matches the Velodyne scan frequency of 10 Hz. Dividing this duration by 2π gives a quantity that describes the time per radiant:

$$timePerRad = \frac{t_{end} - t_{start}}{2\pi}. \#(1)$$

Second, the angle of each point is computed using np.arctan2(y, x). Now, the time duration of when a LiDAR point was registered since the Velodyne was facing forward is

$$\Delta t = -\ angle * timePerRad. \#(2)$$

The minus results from the Velodyne rotation (clockwise) being opposite to the rotation direction of the angle (counter-clockwise). It should be noted that three assumptions were made: (i) a constant rotational speed of the Velodyne (ii) that the image was formed exactly, when the Velodyne was facing forward (angle=0°) and (iii) that the Velodyne started the scan in the backward half of the circle of rotation ($\varphi_{start} > \frac{\pi}{2} or\ \varphi_{start} < -\frac{\pi}{2}$). In other words, the Velodyne starts outside the field of view of the camera.

Assumption (ii) leads to problems that are discussed later (Section "Problems with Assumption (ii)").

Assumption (iii) can be accounted for along the following. In (2), a positive point angle corresponds to a point registered **before** the Velodyne was facing forward. However, if that point angle is greater than the (positive) start angle of the Velodyne, the registration of the point actually happened **after** facing forward. In this case, subtracting 2π from the point's angle would lead to a correct time in (2). This was also implemented in code, but it does not actually have any effect, since the Velodyne always starts the scan facing almost exactly backwards.

The translation of the car in the meantime ΔT can be calculated as $\Delta T = \Delta t * v$, where v ist the velocity reported from the IMU/GPS unit. The Rotation matrix corresponding to the rotational difference can be obtained by using Scipy's rotation implementation $\Delta R =$ Rotation.from_euler('z', $\Delta t * \omega_z$).as_matrix(), where ω_z is the z-component of the angular rates.

A point p can then simply corrected by:

$$\begin{pmatrix} p_{corr} \\ 1 \end{pmatrix} = \begin{bmatrix} \Delta R & \Delta T \\ 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} p \\ 1 \end{pmatrix}$$

Using this method successfully compensates the apparent motion distortion, as shown in FIG 11. Again, it should be noted that an assumption was made, namely that the translation and rotation of the car are independent.

Problems with Assumption (ii): Unfortunately, this method is not always accurate. For frame 312, which is shown in FIG 12 and FIG 13, one can see some shift on the sign in the right image half. Interestingly, this inaccuracy is mitigated when adding the time difference between the image timestamp and the Velodyne forward timestamp to the previous Δt in (2). So, we end up with

$$\Delta t = t_{velo,forward} - t_{image} - angle * timePerRad. \#(3)$$

Using this adaptation, frame 312 is processed correctly, as can be seen in FIG 14. There is a trivial explanation for this result. The time difference between the image timestamp and the Velodyne forward timestamp is about 0.01 seconds, so about 10 % of the Velodyne rotation duration. This translates to an angle offset of $0.1 * 360° = 36°$. The points corresponding to the center of the sign to the right of frame 312 are located at 38˚. Hence, when selecting the image timestamp, the algorithm assumes that the picture was taken right when the points for the sign were recorded and therefore applies almost no correction.

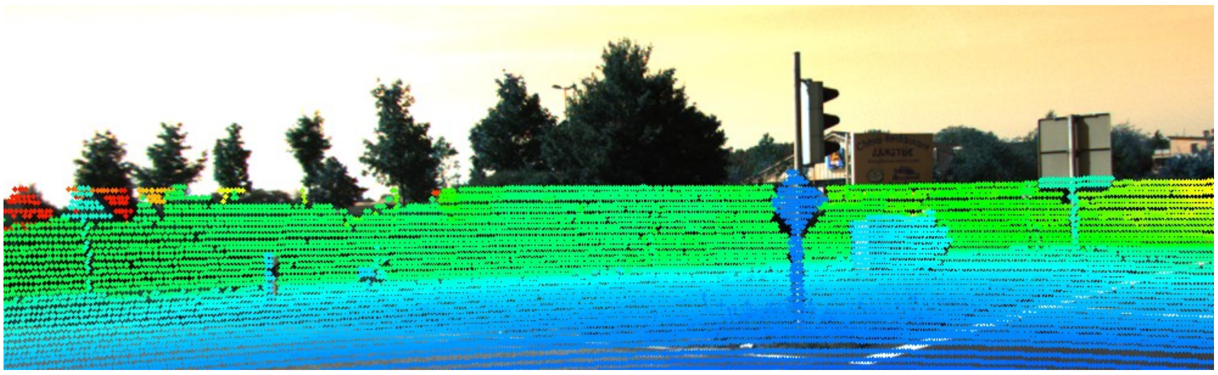


FIG 12. Naïve projection onto frame 312. Surprisingly, there is correction needed.

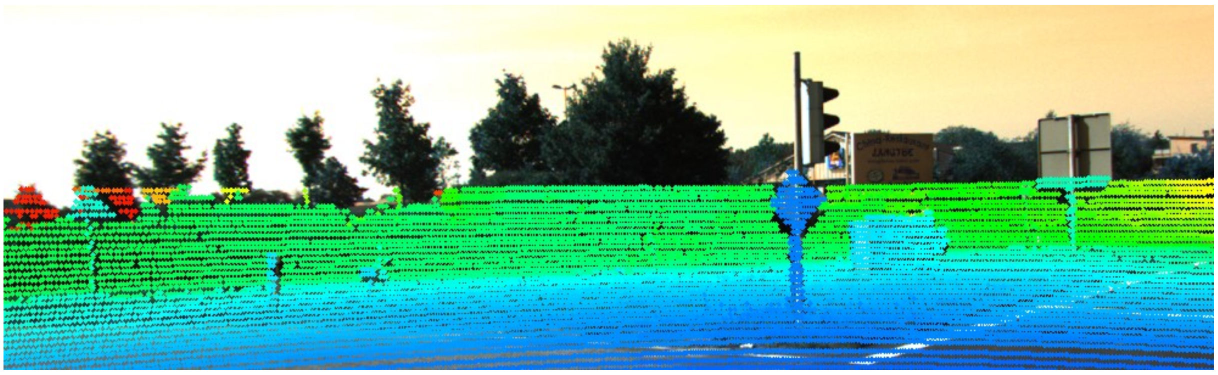


FIG 13. Frame 312 corrected with the Velodyne forward timestamp. There is a shift visible on the right sign.

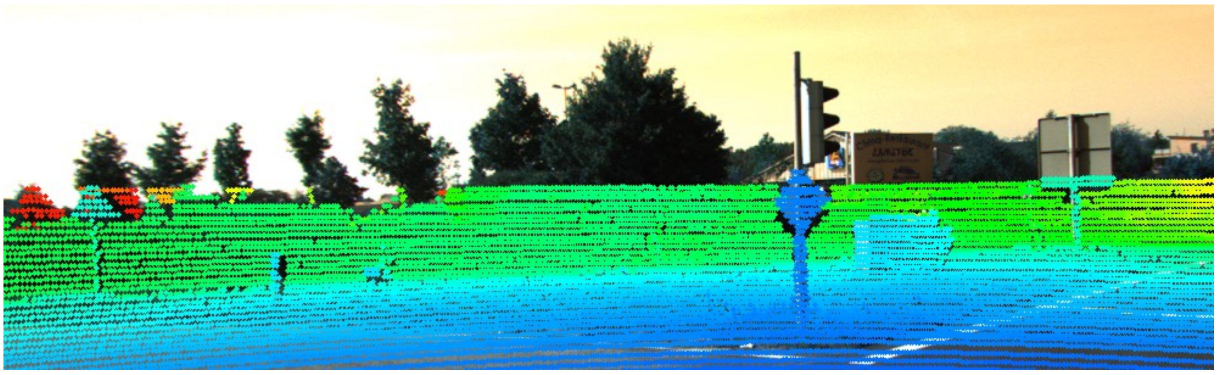


FIG 14. Frame 312 corrected with the image timestamp.

However, using the image timestamp results in an overcorrection for frame 37. From Piazza posts it can be inferred that the true timestamp of image formation is somewhere in between the two provided timestamps and also subject to fluctuation. In conclusion, we believe the data or the assumptions that were made are the source of the observed inaccuracies.

Problem 5. Bonus Questions

1. Eye safety

Infrared lasers mainly inflict thermal damage. Hence the intensity and exposure time are important aspects to assess safety. A LiDAR emits laser pulses radially during operation and in a rotating manner. Because of the radial emission, the density of the point cloud and therefore the density of emitted light pulses in a certain timeframe increases the closer one gets to the LiDAR. At a safe distance, the human eye is only hit by a very small number of pulses. However, the closer one gets, the more light pulses hit the human eye, hence increasing the exposure and likelihood of eye injury. Even though most LiDARs use a class 1 eye-safe laser, this could lead to eye damage, due to the dense and rapid succession of the pulses. Furthermore, interactions of LASER pulses with particles in the atmosphere, such as scattering and refraction, decrease LASER energy. Also, greater distances travelled increase the risk of reduced coherency weakening the signal strength. Therefore, a LASER pulse hitting the human eye at a larger distance from the sensor is less of a medical concern due to weaker energy of the pulses.

2. Wet roads pose challenges for both cameras and LiDAR. What are these challenges and why?

Wet roads become more reflective and show similar characteristics to mirrors. Light emitted by various different sources, e.g. traffic lights and headlights, is strongly reflected by wet surfaces. These reflections are observed by the camera and can lead to false predictions. Furthermore, reflections might obscure road markings which could lead to false decisions. For instance, bright light bands (of different colours) within an image make a correct semantic segmentation more difficult.

As far as LiDAR is concerned, the emitted LASER pulses might “bounce off” the reflective road, leading to an entirely missed detection or the detection of an object further away from its actual target point.

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?

Often the data perceived by the two sensors is not equivalent. E.g. if a stereo sensor pair looks directly at a small cube in front of it, the left sensor can see the left side of the cube, whereas the right side is hidden from it and vice-versa for the right sensor. This also applies to occlusion. A pole in front of some background occludes a different part of the background depending on the viewpoint. In Problem 4, Figure 2, this effect is visible. Even though we corrected for the motion distortion and transformed the points between the coordinate frames, there is a gap in the LiDAR points. This is partially due to the constant difference in viewpoints caused by the offset between the sensors and partially due to the temporal difference between the respective data acquisitions.

The further the sensors are apart, the greater the offset in the viewpoints and therefore the greater the observed differences.