CVL, ETH Zurich | Deep Learning for Autonomous Driving

Project 1: Understanding Multimodal Driving Data

01/03/21 - 21/03/21

General Info: This project accounts for 10% of your grade. You are welcome to ask questions in Piazza. If you think that your question may reveal the solution, please feel free to email your question to: Ozan Unal (ozan.unal@vision.ee.ethz.ch) or Dengxin Dai (dai@vision.ee.ethz.ch)

Overview: In this exercise we will look into understanding multimodal driving data. Firstly, you will create an arsenal of visualisation tools that will help you understand and debug your future models (such as in Project 3). In specific, you will visualize the outputs of common tasks such as 3D object detection and point cloud semantic segmentation given a LiDAR point cloud, corresponding RGB camera image, ground truth semantic labels and network bounding box predictions. Finally you will increase your understanding of the LiDAR sensor itself by identifying each laser ID from the point cloud directly, and dealing with the distortion caused by the vehicle motion with the aid of GPS/IMU data.

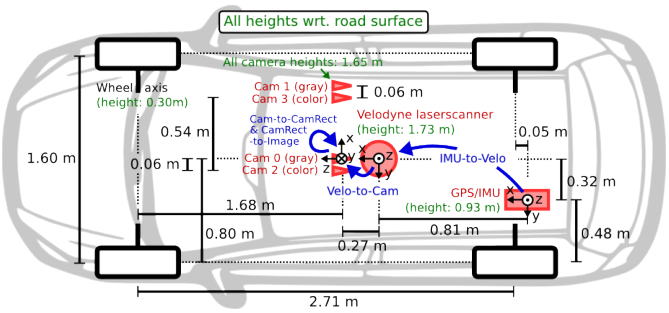


Figure 1: Camera setup.

Materials: In the gitlab repository[1] we provide two autonomous driving scenes that can be used for Problems 1-3. **data.py** must be used for your submission. For each task we provide an example result obtained using **demo.py** which you can use to visually confirm your results. To save you time, the scenes are given as a python dictionary and can be accessed via the provided **load_data.py**. Make sure to install **pickle**.

```
import os
from load_data import load_data

data_path = os.path.join('your/data/dir','data_name.p')
data = load_data(data_path)
```

The data dictionaries provided contain the following keys:

- **velodyne** is the point cloud of the scene. The LiDAR scanner used is the Velodyne HDL-64E that spins at 10 frames per second, capturing approximately 100k points per frame. This sensor has 64 channels. More information about the sensor can be found in the associated data sheet.

[1] https://gitlab.ethz.ch/dlad21/exercise1

The point cloud is given as a (**num_points** x **4**) numpy.array object. The first three dimensions of the array contain the x, y and z coordinates stored in metric (m) using the velodyne coordinate system. The fourth dimension is the reflectance intensity value which is between 0 and 1.

- **image_2** is the RGB image received from left RGB camera (Cam 2 in Fig.1). A capture by the camera is triggered when the velodyne is looking exactly forward.
- **P_rect_X0** gives the intrinsic projection matrices to Cam X after rectification, given as a (**3**x**4**) numpy.array.
  The notation is given as P_rect_destinationFrame-originFrame. Not all of them are needed for the following problems.
- **T_camX_velo** are the homogeneous velodyne to rectified camera coordinate transformations and are provided as (**4**x**4**) numpy.array objects.
  The notation is given as T_destinationFrame_originFrame. Not all of them are needed for the following problems.
- **sem_label** is a (**num_points**,) numpy.array object that gives the semantic label of each point within the scene.
- **color_map** is a dictionary which maps numeric semantic labels to a BGR color for visualization. (Careful, not RGB!)
  Example: 10: [245, 150, 100] # car, blue-ish
- **labels** is a dictionary which maps the numeric semantic labels to a string class.
  Example: 10: "car"
- **objects** contains a list of lists. Each sublist has 16 columns of the following:
  1 **type** describes the type of object. For simplicity we only provide "Car" labels.
  1 **truncated** Float from 0 (non-truncated) to 1 (truncated) refers to the object leaving image boundaries
  1 **occluded** Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded, 3 = unknown
  1 **alpha** Observation angle of object, ranging $[-\pi : \pi]$
  4 **bbox** 2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
  3 **dimensions** 3D object dimensions: height, width, length (in meters)
  3 **location** 3D object location x,y,z in Cam 0 coordinates (in meters) describing the center of the bottom face of the bounding box
  1 **rotation_y** rotation ry around Y-axis in Cam 0 coordinates $[-\pi : \pi]$ centered around its center
  1 **score** Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.
  Example: 'Car', 0.0, 0.0, -1.3474, 266.5866, 192.3474, 444.9974, 312.3929, 1.4426, 1.6142, 4.1894, -3.6577, 1.5748, 11.0496, -1.6602, 0.93]
  For this project you will only need the dimensions, location and rotation_y as all objects are already of the 'Car' type.

The coordinate systems are defined the following way, where directions are informally given from the drivers view, when looking forward onto the road:

- **Velodyne:** x: forward, y: left, z: up
- **Camera:** x: right, y: down, z: forward

For Problem 2.2 we also provide a basic 3D visualization code for point clouds under **3dvis.py** that uses **vispy**.
The data for Problem 4 can be found under the directory **/data/problem_4**. We also provide additional utility functions in **data_utils.py** for Problems 3 and 4.

Submission: For each problem statement, the report (`report.pdf`) should contain a concise description of your solution and the resulting figures. There is no page limit, but please avoid lengthy and redundant descriptions. A corresponding python file for each task (`task1.py`, `task2.py`, `3dvis.py`, `task3.py`, `task4.py`) should be under a **codes** directory. Final reports must be sent by each team in a file named `dlad_ex1_teamID_studentname1_studentname2.zip` to Ozan Unal (ozan.unal@vision.ee.ethz.ch) by 21-03-2021 with subject "`DLAD EX1: YOUR_TEAM_ID`".

Problem 1. Bird's eye view (0.5 point(s))

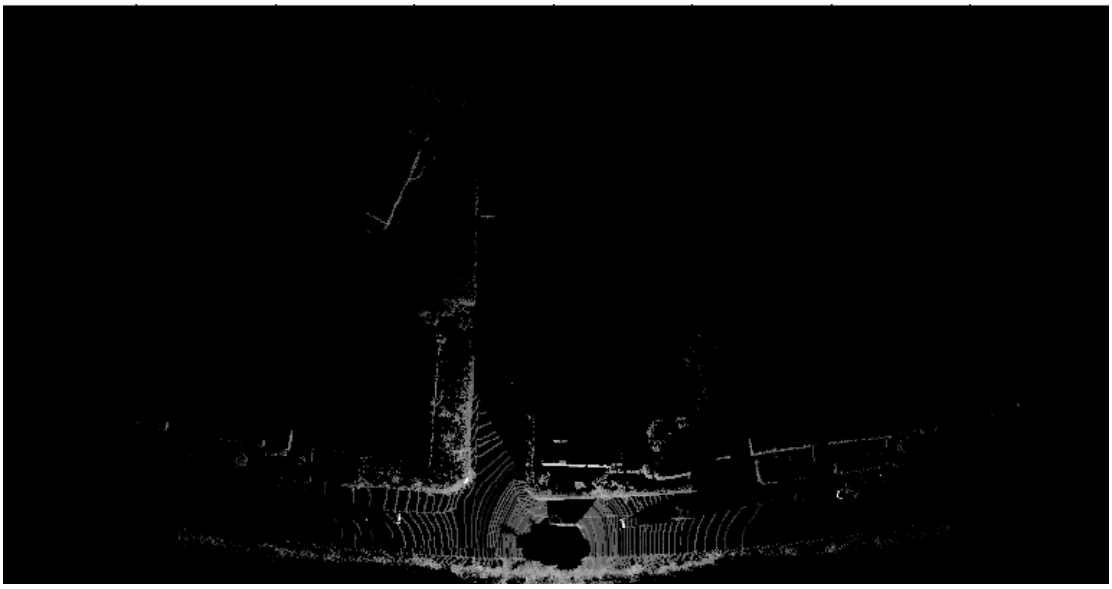


Figure 2: An example of a BEV image with resolution (0.2, 0.2)m in x, y respectively. For visualization purposes the figure is rotated by 90°.

The bird's eye view (BEV) is an elevated view of a scene from directly above. Although BEV brings information loss during projection and discretization, crucially it preserves the metric space. Remember, objects we face in autonomous driving scenes such as cars or pedestrians do not fly (at least not yet)! This allows, detection models to explore priors about the size and shape of the object categories in the BEV view *without* obscurances.

Display the BEV image of the given scene with pixel intensities corresponding to the points' respective reflectance values. The resolution should be 0.2m, 0.2m in x, y coordinates respectively. If multiple points lie within the same bin, the highest intensity point should be sampled. A reference solution can be seen in Fig.2.

Problem 2. Visualization (2+1.5+1=4.5 point(s))

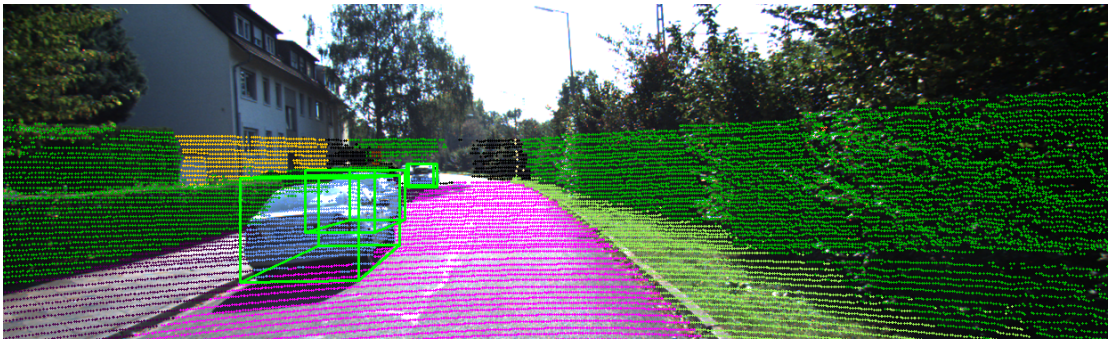


Figure 3: An example of a LiDAR point cloud projected onto the Cam 2 image along with the predicted bounding boxes. The point cloud is colored using **color_map** and the predicted pointwise semantic labels.

We are tasked to create a model that can detect all vehicles within a scene while also generating pointwise semantic labels. After a while we come up with a multitask network that shows promise. Now let's visualize some results from this network. While metrics can tell a compelling story, qualitative assessments also play an important role in understanding a models strengths as well as its weaknesses (keep this in mind for Project 3!). Almost all autonomous driving capable cars that carry a LiDAR scanner run multiple RGB cameras (as they are in comparison much cheaper). However, each measured signal resides within its sensor's coordinate system. Through LiDAR-camera calibration, a rigid transformation matrix is determined to establish correspondences between the points in the 3D space and the pixels in the 2D space.

1. Given the intrinsic and extrinsic projection matrices, project the point cloud onto the image of Cam 2. Color each point projected according to their respective semantic label. The color map for each label is provided with the data. A reference solution can be seen in Fig. 3 that also includes the results from Problem 2.2.

Hint: Make sure to filter your point cloud! The provided Velodyne scans are 360°. Projection equations will give you a result even for points that are behind the camera which will get projected as if they were in front, but vertically mirrored.

Projecting a point cloud onto an RGB image allows us the visually assess the quality of our 3D semantic segmentation results by comparing the estimated labels to the ground truth which can be inferred from the image itself. In addition to the semantic labels of the scene, our network also predicted the 3D bounding boxes of all vehicles. Similar to the segmentation results, the 3D detection results can also be projected onto the camera image to visually assess the quality of the model estimations.

2. In addition to the points with semantic labels, project the 3D bounding boxes of all given vehicles onto the Cam 2 image. A reference solution can be seen in Fig. 3.

Hint: A 3D bounding box is essentially just 8 points, but don't forget to draw the lines in between.

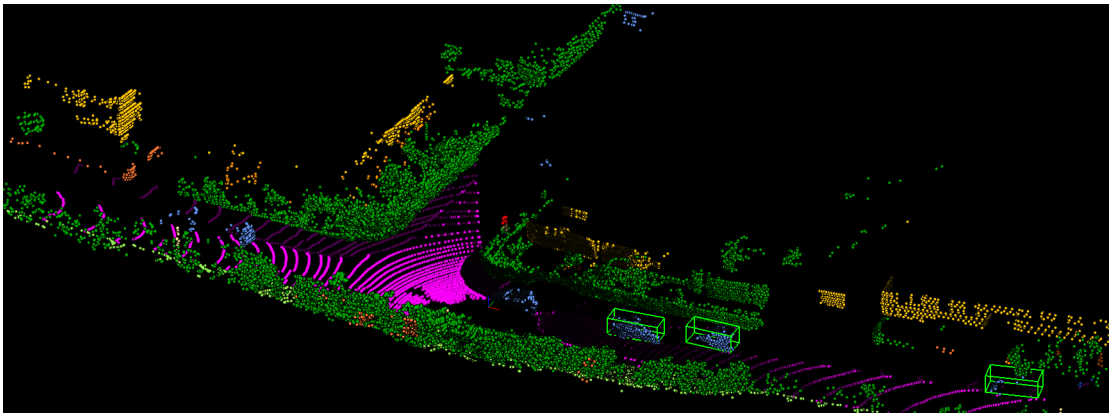


Figure 4: An example of a LiDAR point cloud visualized in 3D using a modified version of the provided **3dvis.py**.

As you might imagine, visualization onto a 2D RGB image is not always ideal. For example, not only is it hard to assess the quality of bounding boxes when projected, but a dataset might contain pointwise semantic labels for a 360° point cloud while only providing a front-facing RGB camera system (as in the case here). It is therefore important to be able to also visualize the point cloud in its native 3D space. To save you the trouble, we provide a bare-bone visualization code **3dvis.py** that can be used to visualize a point cloud as well as 3D bounding boxes.

3. Use and modify **3dvis.py** to visualize the scene in 3D. Color each point according to its semantic label and draw all 3D bounding boxes within the scene. Looking at the results, assuming that the semantic labels are all correct, how many cars did our network fail to identify that lie within the camera field-of-view? A reference solution can be seen in Fig. 4.

Problem 3. ID Laser ID (1 point(s))

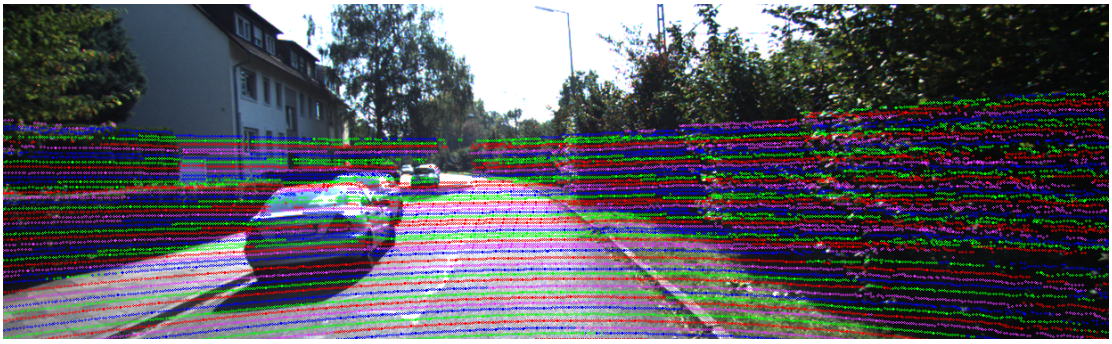


Figure 5: An example of identified Laser IDs.

In this exercise, you need to identify the laser ID of each point (i.e. the point is taken by which of the 64 lasers) in a given 3D point cloud. Laser IDs are normally available in the recorded data, but they can also be "roughly" (due to noise in the data) inferred from the point cloud directly. Some of the information you need about the Velodyne HDL64 LiDAR can be found in Fig. 8. Identify the Laser IDs $i \in \{1, ..., 64\}$ for every point in the point cloud and project them onto Cam 2. You can use four alternating colors to indicate the identified IDs. A reference solution can be seen in Fig. 5.

Problem 4. Remove Motion Distortion (4 point(s))

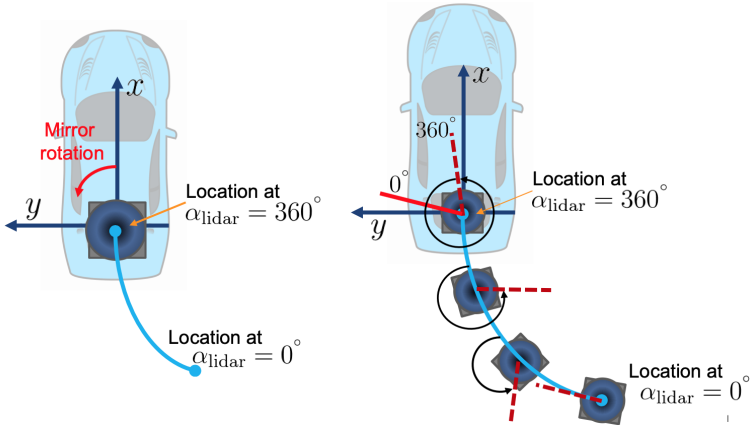


Figure 6: An illustration of where the vehicle's rotation is added to the laser rotation. The LiDAR viewpoint (and measured distances due to vehicle's translation) needs to be "untwisted" according to the vehicle's movement.

As taught in lecture 01, the LiDAR scanner takes depth measurements continuously while rotating around its vertical axis (in contrast to the cameras, which are triggered at a certain point in time). Acquiring data from a moving car means that the car moves fast while the LiDAR rotates. Distortions are caused by the vehicle motion. They depend on the motion speed and the scan period. For instance, our Velodyne HDL64 running at 10 Hz:

- a linear motion at 50 km/h causes a gap of 1.38 m between the begin and the end of a scan.
- a rotation motion at $25°/s$ creates a gap of 2.19 m at a distance of 50 m.

Consequently, the 3D point cloud is distorted and might generate errors if the map is used for precise perception and location based on the LiDAR point cloud. Thus, when computing point clouds we need to 'untwist' the points linearly with respect to the Velodyne scanner's location at the beginning and the end of the 360° sweep. In order to know the locations of the velodyne scanner, we use the data provided by a high-precision combined GPS/IMU system.

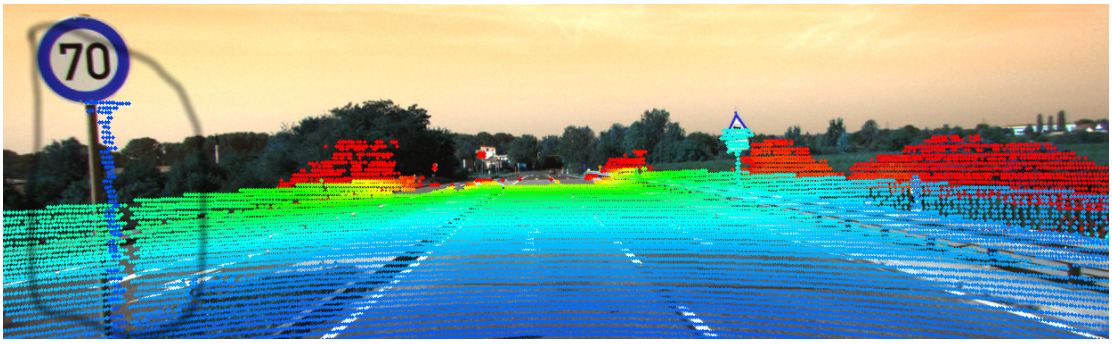
An illustration of the motion distortion can be found in Fig. 6.

We need to work with three sensors in this task: Camera, LiDAR and GPS/IMU. Each sensor stream is stored in a single folder. The main folder contains meta information and a timestamp file, listing the timestamp of each frame of the sequence to nanosecond precision. Numbers in the data stream correspond to each numbers in each other data stream and to line numbers in the timestamp file (0-based index), as all data has been synchronized. The camera has been triggered directly by the Velodyne laser scanner, while from the GPS/IMU system (recording at 100 Hz), we have taken the data information closest to the respective reference frame.

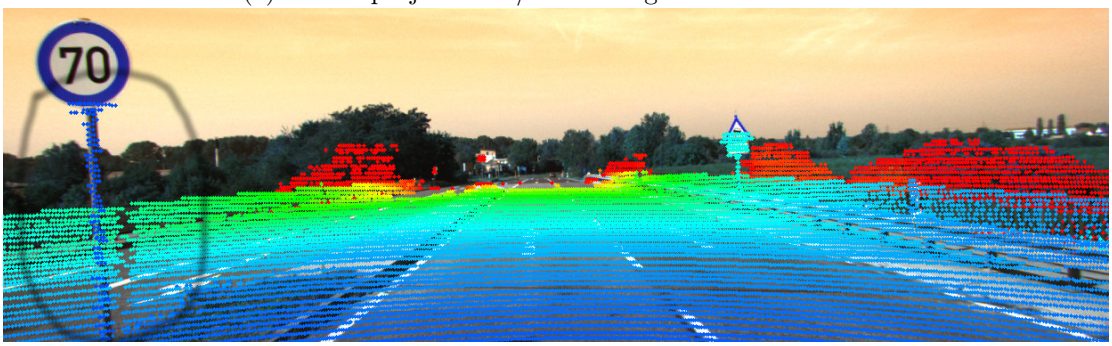
The GPS/IMU information is given in a single small text file which is written for each synchronized frame. All the GPS/IMU files are stored in the folder 'oxts'. Each text file contains 30 values which are listed in Table 1. Not all values are needed for this exercise.

The velodyne point clouds are stored in the folder 'velodyne_points'. To save space, all scans have been stored as Nx4 float matrix into a binary file, where the first 3 values correspond to x, y and z, and the last value is the reflectance information. The time-stamps for the beginning and the end of the sweeps can be found in the timestamps file. The velodyne rotates in clockwise direction. The images are stored in the folder 'image_02', in lossless png format.

A stream of 430 frames are given for this task so that you can see different types of motion. They are in the folder '2011_09_26_drive_0029_sync'. The calibration matrices are provided as



(a): direct projection w/o removing motion distortion



(b): projection after motion distortion removed

Figure 7: An example ('0000000037') showing how motion distortion looks like when projected to an image (a) and how it looks like when corrected (b). Pay attention to the signs.

txt files in the folder as well. For this task, you need to project the LiDAR points to the corresponding images in order to verify the results. Please color the points according to its distance. In your report, please show the results for frame: '0000000037'. You need to show side-by-side the result of a direct projection with motion distortion and the result when motion distortion is removed. A reference solution can be found in Fig. 7. Some of the functions you might need are provided in **'data_utils.py'**.

Hint: For a specific frame, the time order is: 'LiDAR_starts' → 'camera_triggered' → 'LiDAR_ends'. You need to use these timestamps to decide where (relative to the front view, i.e. the camera direction) the LiDAR starts the scan for each frame. Remember that all cameras have been triggered directly by the Velodyne laser scanner; the relationship between the camera triggers and the velodyne is the following: the cameras are triggered when the velodyne is looking exactly forward (into the direction of the cameras).

Hint: In the provided LiDAR point cloud, the order of the points is lost – no information about which point is taken before which other points. You need to figure this out based on where the LiDAR starts each scan. Since the lasers fire in sequence, we can figure out the order, from the first point to the last point, by ranking the points according to one value which you need to calculate.

Hint: Since short term road vehicle motion is mostly planar, you can only consider the rotation around z axis, i.e. the yaw.

Hint: The untwisted point cloud can be understood as if it were generated when the vehicle is static at a location. This location needs to be specified via timestamp.

Hint: To keep it simple, you can assume that the translation and rotation are independent, and they can be modeled by separate linear models. The whole task can be solved as an interpolated rigid transformation problem.

HDL-64E

High Definition Lidar Sensor

The HDL-64E S3 provides high definition 3 dimensional information about the surrounding environment.

	Specifications:
Sensor:	• 64 channels • Measurement Range: Up to 120 m • Range Accuracy: Up to ±2 cm (Typical)[1] • Field of View (Vertical): +2.0° to -24.9° (26.9°) • Angular Resolution (Vertical): 0.4° • Field of View (Horizontal): 360° • Angular Resolution (Horizontal/Azimuth): 0.08° – 0.35° • Rotation Rate: 5 Hz – 20 Hz
Laser:	• Laser Product Classification: Class 1 Eye-safe • Wavelength: 903 nm
Mechanical/ Electrical/ Operational	• Power Consumption: 60 W (Typical)[2] • Operating Voltage: 12 V – 32 V • Weight: 28 lbs. (12.7 Kg) (without cabling) • Dimensions: 215 mm Diameter x 283 mm Height (Base: 203 mm x 203 mm) • Operating Temperature: -10°C to +60°C[3] • Storage Temperature: -40°C to +85°C
Output:	• 3D Lida r Data Points Generated: - Single Return Mode: ~1,300,000 points per second - Dual Return Mode: ~2,200,000 points per second[4] • 100 Mbps Ethernet Connection • UDP Packets Contain: - Time of Flight Distance Measurement - Intensity Measurement - Rotation Angles - Synchronized Time Stamps (µs resolution) • GPS: $GPRMC NMEA Sentence from GPS Receiver (GPS not included)

63-9194 Rev-K

For more details and ordering information, contact Velodyne Sales (sales@velodyne.com)

1. Greater than or equal to 80% of channels at ambient wall test; remaining channels better than or equal to 5 cm.
2. Operating power may be affected by factors including but not limited to range, reflectivity and environmental conditions.
3. Operating temperature may be affected by factors including but not limited to air flow and sun load.
4. Configuration dependent.

CLASS 1 LASER PRODUCT

Velodyne Lidar, Inc. 5521 Hellyer Ave, San Jose, CA 95138 / lidar@velodyne.com / 408.465.2800 velodynelidar.com

Figure 8: The Specifics of Velodyne HDL64 S3.

1	- lat:	latitude of the oxts-unit (deg)
2	- lon:	longitude of the oxts-unit (deg)
3	- alt:	altitude of the oxts-unit (m)
4	- roll:	roll angle (rad), 0 = level, positive = left side up, range: -pi .. +pi
5	- pitch:	pitch angle (rad), 0 = level, positive = front down, range: -pi/2 .. +pi/2
6	- yaw:	heading (rad), 0 = east, positive = counter clockwise, range: -pi .. +pi
7	- vn:	velocity towards north (m/s)
8	- ve:	velocity towards east (m/s)
9	- vf:	forward velocity, i.e. parallel to earth-surface (m/s)
10	- vl:	leftward velocity, i.e. parallel to earth-surface (m/s)
11	- vu:	upward velocity, i.e. perpendicular to earth-surface (m/s)
12	- ax:	acceleration in x, i.e. in direction of vehicle front (m/s^2)
13	- ay:	acceleration in y, i.e. in direction of vehicle left (m/s^2)
14	- az:	acceleration in z, i.e. in direction of vehicle top (m/s^2)
15	- af:	forward acceleration (m/s^2)
16	- al:	leftward acceleration (m/s^2)
17	- au:	upward acceleration (m/s^2)
18	- wx:	angular rate around x (rad/s)
19	- wy:	angular rate around y (rad/s)
20	- wz:	angular rate around z (rad/s)
21	- wf:	angular rate around forward axis (rad/s)
22	- wl:	angular rate around leftward axis (rad/s)
23	- wu:	angular rate around upward axis (rad/s)
24	- posacc:	velocity accuracy (north/east in m)
25	- velacc:	velocity accuracy (north/east in m/s)
26	- navstat:	navigation status
27	- numsats:	number of satellites tracked by primary GPS receiver
28	- posmode:	position mode of primary GPS receiver
29	- velmode:	velocity mode of primary GPS receiver
30	- orimode:	orientation mode of primary GPS receiver

Table 1: The 30 values (in order) from the GPS/IMU system.

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

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

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

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?