CVL, ETH Zurich | Deep Learning for Autonomous Driving

Project 1: Understanding Multimodal Driving Data

28/02/20 - 20/03/20

Overview: Successfully completing all projects is compulsory for attending the exam. The projects can be completed in groups of 2. This project will affect your course grade by 10%. For further questions contact: Ozan Unal (ouenal@ethz.ch) or Dengxin Dai (dai@vision.ee.ethz.ch).

Submission: For each problem statement, the report should contain a concise description of your solution. You should also include the resulting figure and indicate the code files for the task. Submissions must be made in a ZIP file to Ozan Unal (ouenal@ethz.ch) by 20 March 2020. The ZIP file should include a report (PDF) for your answers and your Python code. The code will be checked by us.

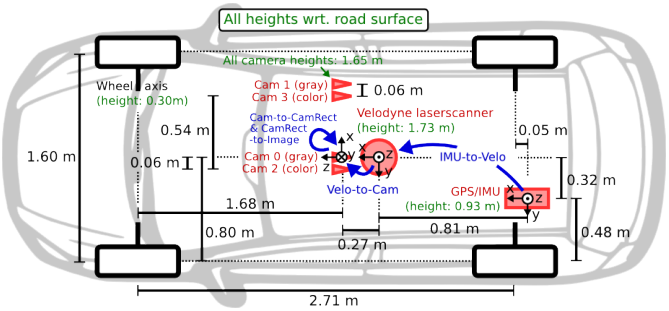


Figure 1: Camera setup.

Materials: In the gitlab repository[1] available to you are two autonomous driving scenes. 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**.

```
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.

  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.

[1] https://gitlab.ethz.ch/ouenal/deep-learning-for-autonomous-driving.git

- **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.

- **K_camX** provide the pinhole camera intrinsics from Cam 0 to Cam X and are given as (**3**x**3**) numpy.array objects.

Only for **segmentation_data.p** the following *additional* keys can be found:

- **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.

- **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.
  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"

Only for **detection_data.p** the following *additional* key can be found:

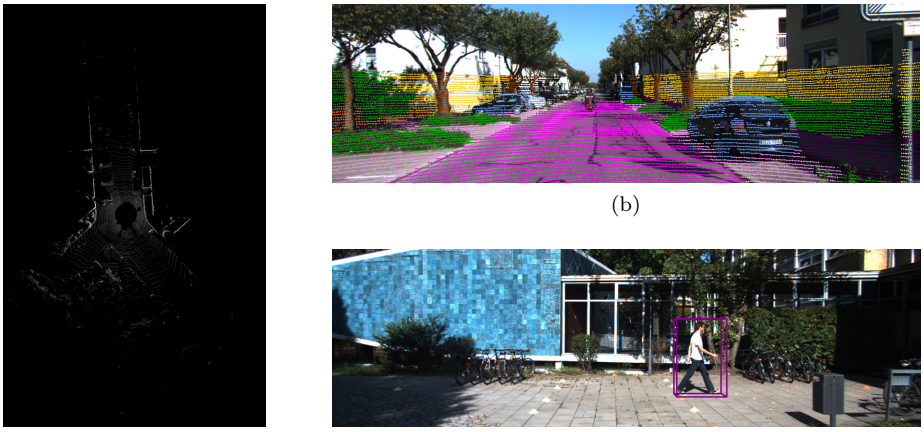
- **objects** contains a list of lists. Each sublist has 8 columns of the following:
  1 **type** describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
  3 **dimensions** 3D object dimensions: height, width, length (in meters)
  3 **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 $[-\pi : \pi]$
  Example: [Car, 1.65, 1.67, 3.64, -0.65, 1.71, 46.70, -1.59]

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

An additional **data_utils.py** is provided for **Problems 3-4**. The data files of **Problem 4** can be found under the directory **'/data/problem_4'**.



(a) (b) (c)

Figure 2: Example results for problems 1-2. (a) BEV image, (b) Point cloud projected onto image and colored according to semantic labels, (c) 3D bounding box projected onto the image.

Problem 1. Bird's Eye View (1.5 point(s))

Fun fact: 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 in **segmentation_data.p** with pixel intensities corresponding to the points' respective reflectance values. A reference solution can be seen in Fig.2a where the projected point cloud is discretized into a 2D grid with resolution of 0.2m, 0.2m and 0.3m in x, y and z coordinates respectively.
Tip: Having problems visualizing your data? Maybe today is the day you learn[2] about cv2!

Problem 2. Projection onto Image Plane (2.5+1=3.5 point(s))

a. Semantic Segmentation: Displaying Semantic Labels

Project the point cloud of **segmentation_data.p** onto the image of Cam 2. Color each point projected according to their respective label. The color map for each label is provided with the data. A reference solution can be seen in Fig.2b.
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.

b. Object Detection: Drawing 3D Bounding Boxes

It's time for object detection! Project the 3d bounding boxes of all given vehicles, cyclists and pedestrians within the scene of **detection_data.p** to the Cam 2 image. A reference solution can be seen in Fig.2c.
Hint: A 3D bounding box is essentially just 8 points!

[2] https://opencv-python-tutroals.readthedocs.io/

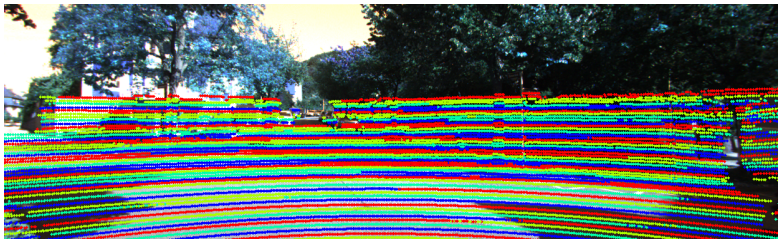


Figure 3: An example of identified Laser IDs.

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

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) figured out from the point cloud directly. Some of the information you need about the Velodyne HDL64 LiDAR can be found in Fig. 7. Please identify the Laser IDs $i \in \{1, ..., 64\}$ for every point in the point cloud in **segmentation_data.p** and project them to Cam2. You can use four alternating colors to indicate the identified IDs. A reference solution can be seen in Fig.3.

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

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. 4.

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

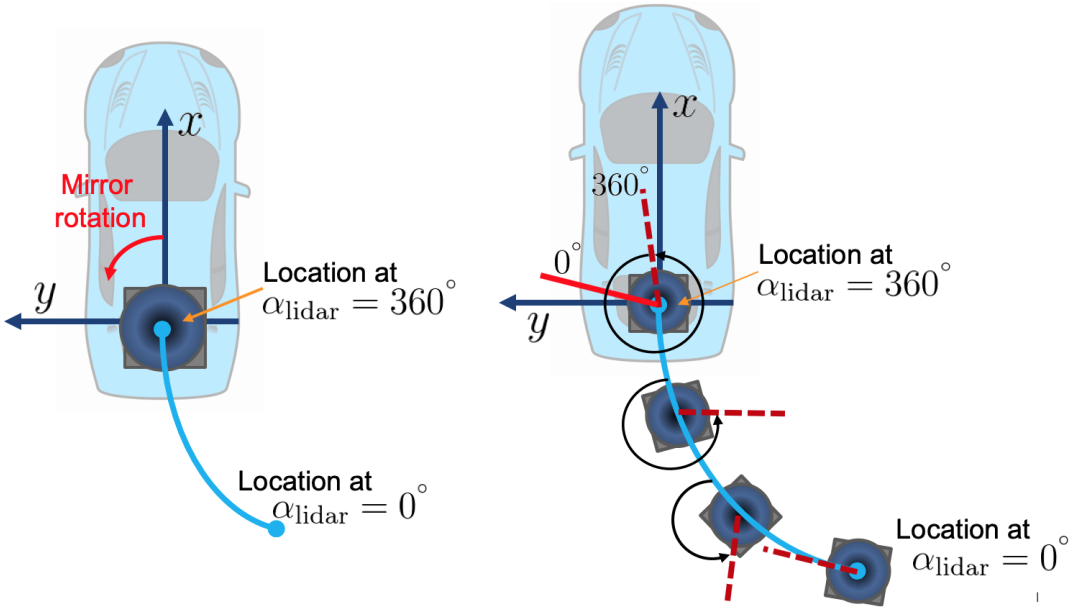


Figure 4: An illustration that the vehicle's rotation is added to the laser rotation. The LiDAR viewpoint (and measured distances due to vehicle's translation) needs to be 'untwisted' according to the vehicle's movement.

and the end of the sweeps can be found in the timestamps file. The velodyne rotates in counter-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 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 three frames: '0000000037', '0000000310' and '0000000320'. You need to show side-by-side the result of a direct projection with motion distortion and the result when motion distortion is removed. Reference solutions can be found in Fig. 5 and Fig. 6. 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).

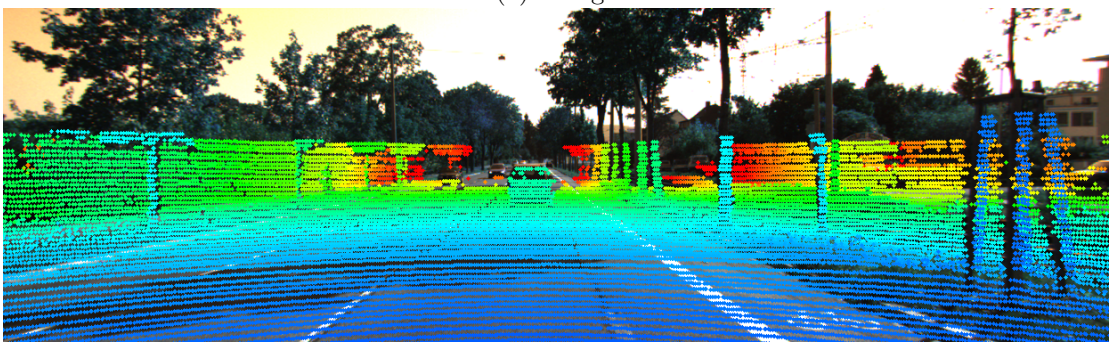
Hint: In the provided Lidar point cloud, the order of the points is lost – no information about which point is taken before which other points. You need to figure this out based on where the LiDAR starts each scan. Since the lasers fire in sequence, we can figure out the order, from the first point to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

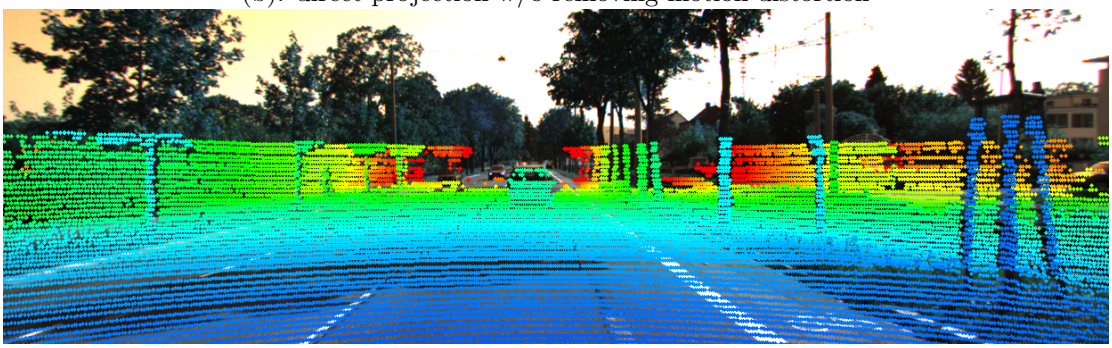
Hint: The untwisted point cloud can be understood as if it were generated when the vehicle is static at a location. This location needs to be specified via timestamp.

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.

(a): image



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



(c): projection after motion distortion removed

Figure 5: An example showing how motion distortion looks like when projected to an image (b) and how it looks like when corrected (c). Pay attention to the tree guide grids.

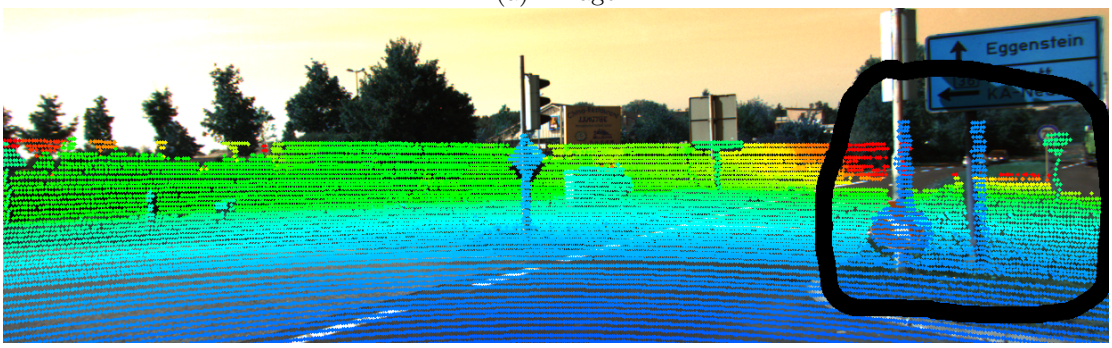
Problem 5. Bonus Questions: you can earn 1 more point (1 point(s))

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

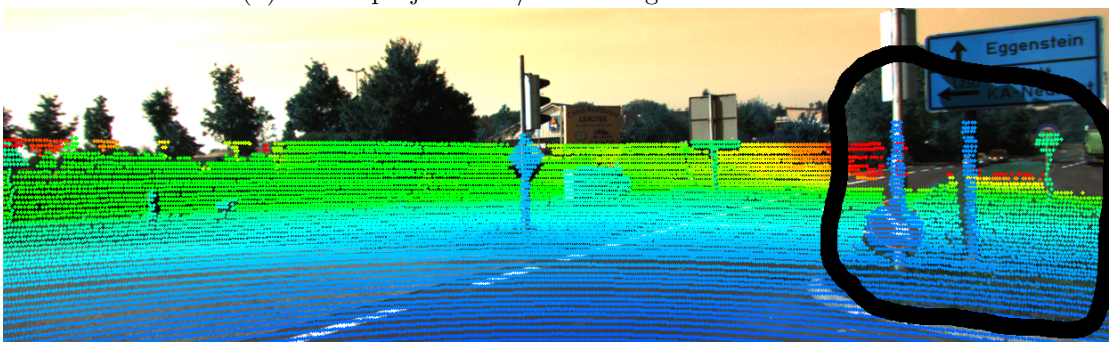
2. Wet road poses challenges for both cameras and LiDAR, what are the challenges and why?

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?

(a): image



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



(c): projection after motion distortion removed

Figure 6: An example ('0000000310') showing how motion distortion looks like when projected to an image (b) and how it looks like when corrected (c). Pay attention to the signs.

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 7: 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.