2) Understanding data

Time consideration

During acquisition, data are timestamped at their arrival on the computer. This time *Rtime* (for Reception time) is expressed in microsecondes from the beginning of the acquisition. The acquisition started a the UTC time *starting_time* (time expressed in microsecondes).

The UTC acquisition time can be obtained as :

$$AcquisitionTimeUTC = \underbrace{Rtime + latency}_{Acquisition time\ (relative\ reference)} + starting\_time$$

The latency is unknown for our sensors and supposed to be null.

Due to the huge amount of data, two computers (PC1 and PC2) are required to enable a real-time lossless storage. In order to perform a consistent timestamping of the sensor readings across the two computers, a time synchronization is performed using the Network Time Protocol (NTP): the clock of PC2 (NTP client) is registered to the clock of PC1 (NTP server) with a bias before correction under half a millisecond. This bias can be supposed to be null.

The data of each interface are saved in a folder (one interface *ID_INTERFACE* by visual sensor, range sensor and GPS, one interface for the CAN bus where data from the odometry, accelerometer and gyrometer are available)

ID_INTERFACE

The *ID_INTERFACE* for the different sensors are the following :

Sensor	Alias	*ID_INTERFACE*
Visual sensors		
Forward left camera	*f-l-cam*	Bus_InterfaceCamera_2672909685359666
Forward right camera	*f-r-cam*	Bus_InterfaceCamera_2672909685359667
Backward camera	*b-cam*	Bus_InterfaceCamera_2672909685359670
Forward middle camera	*f-m-cam*	Bus_InterfaceCamera_2892819404705840
Catadioptric camera	*cata-cam*	Bus_InterfaceCamera_13602058368439990
Fisheye camera	*fe-cam*	Bus_InterfaceCamera_13602058368474999
Webcam	*web-cam*	Bus_InterfaceCamera__dev_video0
2D Range sensors		
Horizontal laser	*hz-laser*	Bus_InterfaceRangefinder_172_27_30_21_2112
Inclined laser	*incl-laser*	Bus_InterfaceRangefinder_172_27_30_31_2112
GPS sensors		

RTK GPS	*rtk-gps*	Bus_InterfaceGps__dev_ttyM0
Low-cost GPS	*lc-gps*	Bus_InterfaceGps__dev_ttyACM0
Proprioceptive sensors		
		Bus_InterfaceCan_can0

(alias are used within the IPDSLib)

Log formats

VISUAL SENSORS

Images and timestamp

The file *ID_INTERFACE*.dates contains data about the acquisition timestamp while the data of i^{th} image is saved in the file *ID_INTERFACE*-*num*.*extension*.

Log format for visual sensors

line	***ID_INTERFACE*.dates**				Associated files
#1 (header)	Version				
	Rtime	*Rtime*+ *latency*	*latency*	→	*ID_INTERFACE*-*num*.*extension*

Details:

- *Rtime* is the reception time expressed in microseconds from the beginning of the acquisition
- *latency* is the camera latency, expressed in microseconds
- the image number is the line number plus one, with 10 digits length
- the file *extension* is pgm (case of the firewire camera when images are in gray-level) or ppm (case of the color cameras)

Sample

Sample for camera of id *ID_INTERFACE* =Bus_InterfaceCamera_2672909685359666 (Forward left camera ; extension=pgm):

Bus_InterfaceCamera_2672909685359666.dates		Associated files
Version 1 121558 121558 0	→	Bus_InterfaceCamera_2672909685359666-0000000001.pgm 
255049 255049 0	→	Bus_InterfaceCamera_2672909685359666-0000000002.pgm 
388331 388331 0	→	Bus_InterfaceCamera_2672909685359666-0000000003.pgm 

RANGE SENSORS

Range sensor impacts and timestamp

The file *ID_INTERFACE*.dates contains data about the acquisition timestamp while the data of i^{th} scan is saved in the file(s) *ID_INTERFACE*-*num*-*numlay*.txt.

Log format for range sensors

line	***ID_INTERFACE*.dates**				Associated files
#1 (header)	Version XXX				
	Rtime	Rtime+ latency	latency	→	*ID_INTERFACE*-*num*-*numlay*.txt

line	***ID_INTERFACE*-*num*-*numlay*.txt**	
#1 (header)	*numimpacts*	
	angle	*distance*

Details:

- Rtime is the reception time expressed in microseconds from the beginning of the acquisition
- latency is the range sensor latency, expressed in microseconds
- numlay is the number of the layer. LMS151 sensors have only one layer.
- numimpacts is the number of impacts
- angle is the angle of the scan, expressed in radians
- distance is the distance to the impact, expressed in meters. A distance of zero means that there is no impact in the range of distance of the sensor.

Sample

Sample for range sensor of id *ID_INTERFACE* =Bus_InterfaceRangefinder_172_27_30_21_2112 (a single layer numlay=0):

Bus_InterfaceRangefinder_172_27_30_21_2112.dates	Associated files
Version XXX 191 193478 213493	 → Bus_InterfaceRangefinder_172_27_30_21_2112-1-0.txt → Bus_InterfaceRangefinder_172_27_30_21_2112-2-0.txt → Bus_InterfaceRangefinder_172_27_30_21_2112-3-0.txt

Bus_...-1-0.txt	**Bus_...-2-0.txt**	**Bus_...-3-0.txt**
541 -0.785398 0.135 -0.776672 0.115 -0.767945 0.113 -0.759218 0.092	541 -0.785398 0.14 -0.776672 0.126 -0.767945 0.112 -0.759218 0.113	541 -0.785398 0.115 -0.776672 0.127 -0.767945 0.123 -0.759218 0.107

GPS sensors

3D location and accuracy data

These data are provided by the essential fix data from GGA sentence (refer to http://www.gpsinformation.org/dale/nmea.htm#GGA). These data have been processed to give a file (*ID_INTERFACE*_GGA_all.txt) which contains the relevant information for most of the applications.

Log format for GPS sensors

line	**_ID_INTERFACE__GGA_all.txt**											
	Rtime	*lat_l2e*	*lng_l2e*	*alt_l2e*	*lat_wgs*	*lng_wgs*	*alt_wgs*	*nb_sat*	*utc_time*	*fix_quality*	*HDOP*	*age_corr_diff*

Details:

- *Rtime* is the reception time expressed in microseconds from the beginning of the acquisition
- Two coordinates are used for GPS positions: WGS84 (*_wgs*) and Lambert2IIe (*_l2e*). *lat_l2e* and *lng_l2e* are the latitude and longitude, expressed in radians for WGS84 coordinates and in meters for Lambert2IIe coordinates. *alt_l2e* is the sum of the altitude above mean sea level (ellipsoidal height) and the height of geoid above ellipsoid, expressed in meters. In our experiments, the height of geoid (mean sea level) above WGS84 ellipsoid is equal to 50.090 for the RTK-GPS and 47.4 for the UBlox.
- *nb_sat* is the number of satellites being tracked
- *utc_time* is the UTC time when data is taken, expressed in the format YYYYMMDD (as in the GGA sentence).
- *fix_quality* is the fix quality (2 = DGPS fix for the UBlox and 4 = Real Time Kinematic for the RTK-GPS)

- *HDOP* is the horizontal dilution of position
- *age_corr_diff* is the time in seconds since last DGPS update

Sample

Sample for GPS sensor of id *ID_INTERFACE*=Bus_InterfaceGps__dev_ttyACM0:

Bus_InterfaceGps__dev_ttyACM0_GGA_all.txt
476121 660168.799340458 2084722.75496854 460.600012207031 0.798665417515059 0.0542784685555518 460.600012207031 8 32322 2 0.980000019073486 -1
1475353 660168.786374253 2084722.75484107 460.600012207031 0.798665417515059 0.0542784656466697 460.600012207031 8 32323 2 0.980000019073486 -1
2473341 660168.786191338 2084722.77336542 460.699987792969 0.798665420423941 0.0542784656466697 460.699987792969 8 32324 2 1.25 -1

Matlab code

```matlab
% Class
classdef gps_data
	properties (Access=public)
		lat_l2e;
		lng_l2e;
		alt_l2e;
		lat_wgs;
		lng_wgs;
		alt_wgs;
		nb_sat;
		utc_time;
		fix_quality;
		HDOP;
		age_corr_diff;
	end
	methods
		function o=gps_data(starting_time,varargin)
			if(nargin==11)
				o.lat_l2e= varargin{1};
				o.lng_l2e= varargin{2};
				o.alt_l2e= varargin{3};
				o.lat_wgs= varargin{4};
				o.lng_wgs= varargin{5};
				o.alt_wgs= varargin{6};
				o.nb_sat= varargin{7};
				o.utc_time= varargin{8};
				o.fix_quality= varargin{9};
				o.HDOP= varargin{10};
				o.age_corr_diff= varargin{11};
			elseif(nargin==2)
				o.lat_l2e= varargin{1}(1);
				o.lng_l2e= varargin{1}(2);
				o.alt_l2e= varargin{1}(3);
				o.lat_wgs= varargin{1}(4);
				o.lng_wgs= varargin{1}(5);
				o.alt_wgs= varargin{1}(6);
				o.nb_sat= varargin{1}(7);
				o.utc_time= varargin{1}(8);
				o.fix_quality= varargin{1}(9);
				o.HDOP= varargin{1}(10);
				o.age_corr_diff= varargin{1}(11);
			end
		end
	end
end

% code to obtain the 10th GPS GGA data as an object
starting_time= importdata("starting_time_us.txt") ;
data = importdata("Bus_InterfaceGps__dev_ttyACM0_GGA_all.txt"," ",0) ;
data_num_10=gps_data(starting_time,data(10,1:end)) ;
```

Text 1: Matlab code to import GGA data

Tracks files

Additionnaly, we provide the trajectory in GPX and KML formats (track) in the files *ID_INTERFACE*_GPX.gpx and *ID_INTERFACE* _KML.kml.

For the GPX file, the schema version 1.1 is used (http://www.topografix.com/GPX/1/1/). In *ID_INTERFACE*_GPX.gpx, the GPS data are encoded as a track (an ordered collection of points). For each trackpoint, the following tags are filled: latitude and longitude (in decimal degrees, WGS84 datum), elevation, fix, hdop, sat, ageofdgpsdata and time are filled. The trajectory has been converted to the KML format (format of the Google file used to display geographic data in Google Earth and Google Map) using GPX to KML online converter.

Sample

Sample for GPS sensor of id *ID_INTERFACE*=Bus_InterfaceGps__dev_ttyACM0:

Bus_InterfaceGps__dev_ttyACM0_GPX.gpx

```
<?xml version="1.0" ...
...
<trk>
  <trkpt lat="45.76015767" lon="3.109927167">
    <ele>460.6000122</ele>
    <fix>2</fix>
    <sat>8</sat>
    <hdop>0.9800000191</hdop>
    <ageofdgpsdata>-1</ageofdgpsdata>
  </trkpt>
...
</xml>
```

Satellites in View

Data about the satellites that the unit might be able to find are extracted from the GSV sentences (refer to http://www.gpsinformation.org/dale/nmea.htm#GSV) and are given in the file *ID_INTERFACE*_GSV.txt.

line

***ID_INTERFACE*_GSV.txt**								
Rtime	*nb_sat*	*nb_sat_data*	*idcanal_1*	*el_1*	*az_1*	*SNR_1*	...	*SNR_12*

Details:

- Rtime is the reception time expressed in microseconds from the beginning of the acquisition
- *nb_sat_data* is the number of satellites with available GSV data (under n*b_sat satelittes)*
- idcanal_1 is the satellite PRN number
- el_1 is the elevation angle, expressed in degrees
- az_1 is the azimuth angle, expressed in degrees
- SNR_1 is the Signal to Noise Ratio
- idcanal, el, az and SNR are given for each (possible) 12 satellites.

Sample

Sample for GPS sensor of id *ID_INTERFACE*=Bus_InterfaceGps__dev_ttyACM0:

Bus_InterfaceGps__dev_ttyACM0_GSV.txt
476606 11 3 2 0.453785598278 0.907571196556 40 12 0.558505356312 1.53588974476 39 14 0.331612557173 3.92699074745 38 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
476850 11 3 25 1.20427715778 1.1344640255 40 29 1.46607661247 3.35103225708 40 30 0.209439516068 5.02654838562 25 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
476975 11 3 33 0.593411922455 3.57792496681 35 39 0.575958669186 2.61799383163 34 40 0.296705961227 2.07694172859 65535 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

NMEA sentences

For persons that are accustomed to processing directly the NMEA sentences, those sentences are listed in the file *ID_INTERFACE*.txt and can be accessed with their acquisition time using the file *ID_INTERFACE*.dates.

line	***ID_INTERFACE*.dates**				
#1 (header)	Version XXX				
	sentence_type	*numchar*	*Rtime'*	*Rtime*'+*latency*'	*latency'*

Details:

- sentence_type is the type of sentence (i.e. GGA or GSV in our acquisitions)
- numchar is the number of characters in the file .txt until the sentence is printed
- Rtime' is the reception time expressed in hour:minute::second.microseconds from the beginning of the acquisition
- latency' is the latency, expressed in hour:minute::second.microseconds

Proprioceptive sensors

Multiple proprioceptive sensors are put on the vehicle: motor encoder, steergin angle encoder, right and left wheels encoders, accelerometers and a magnetometer. For those sensors, both estimated data (using estimated parameters) and raw data are provided.

Motor encoder

line	***ID_INTERFACE*_Motor.txt**						
	Rtime	*dt_odo_motor*	*transl_vel*	*distance_odo*	*direction*	*raw_speed*	*raw_odom*

Details:

- *transl_vel* is the translationnal velocity, expressed in m/s
- distance_odo is the estimated distance travelled by the robot, expressed in m

- These elements are computed using the following formula:

$$transl\_vel = kspeed.rawspeed$$
$$steering\_angle = kangle.rawsteering\_angle$$
$$distance\_odo = kdistance\_odo.odom$$

According to the datasheet, the speed encoder has a resolution of 0.001 m/s thus we set kspeed = 1./(0.001).

$$kdistance\_odo = \frac{\pi.wheel\_diameter}{64}.reductionratio$$

(64 is the number of tops by revolution)

The diameter of the wheel (wheel_diameter) is 49cm (but depends on the pressure of the wheel), the motor_ratio is 1/9.9 thus the travelled distance is k_distance_odo=0.0024 m.

- `direction` is the direction of the vehicle:
    - 0: the vehicle is at rest
    - 1: the vehicle is moving forward
    - 2: the vehicle is moving backward

- raw_speed is the speed measured by the motor odometry. We supposed that the translationnal velocity is proportionnal to the raw value:

$$trans\_vel = kspeed.raw\_speed$$

The value of the parameter according to the datasheet is: kspeed=1./(0.001)(resolution 0.001 m/s)

For our sensor, we estimate those values to be:
kspeed 1095.

- is the value of the motor encoder counter. We supposed that the distance travelled by the vehicle is proportionnal to the raw value.

The value of the parameter according to the datasheet is: kodom=1./(0.001)(resolution 0.001 m/s)The sensor gives us 128 "top" per motor revolution, and behind one reducer of 1:9.9 and with wheels about 40 cm in diameter, this represents approximately 1mm per "top" (0.40 * 3.14/9.9/128) - Specific calibration datas (see [calibration]) helped us to specify the value of 1.095 mm/top

Matlab

```
% Class
classdef odom_motor_data
        properties (Access=public)
                dt_odo_motor;
                transl_vel;
                distance_odo;
                raw_speed;
                raw_odom;
        end
        methods
                function o=odom_motor_data(starting_time,varargin)
                        if(nargin==5)
                                o.dt_odo_motor= varargin{1};
                                o.transl_vel= varargin{2};
                                o.distance_odo= varargin{3};
                                o.raw_speed= varargin{4};
                                o.raw_odom= varargin{5};
                        elseif(nargin==2)
                                o.dt_odo_motor= varargin{1}(1);
                                o.transl_vel= varargin{1}(2);
                                o.distance_odo= varargin{1}(3);
                                o.raw_speed= varargin{1}(4);
                                o.raw_odom= varargin{1}(5);
                        end
                end
        end
end

% code to obtain the 10th motor's odometry data as an object
starting_time= importdata("starting_time_us.txt") ;
data = importdata("Bus_InterfaceCan_can0_Motor.txt"," ",0) ;
data_num_10=odom_motor_data(starting_time,data(10,1:end)) ;
```

Text 2: Matlab code to import odometry data (at the motor level)

Steering angle encoder

line	***ID_INTERFACE*_Steering.txt**		
	Rtime	*steering_angle*	*raw_steering_angle*

Details:

- *steering_angle* is the steering angle, expressed in radians
- raw_steering_angle is the steering angle encoder value (int16_t). We supposed that the steering angle is a linear function of the raw value:

  $$steering\_angle = kangle.raw\_steering\_angle + offset\_angle$$

  The values of the parameters according to the datasheet are: kangle=1./(60.*π/180.0/32768.)(-60deg < steering_angle (deg) < +60deg; resolution of 0.002deg) and offset_angle=0.

  For our sensor, we estimate those values to be:
  kangle 31060. offset_angle -0.004

Matlab

```
% Class
classdef steering_data
        properties (Access=public)
                steering_angle;
                raw_steering_angle;
        end
        methods
                function o=steering_data(starting_time,varargin)
                        if(nargin==2)
                                o.steering_angle= varargin{1};
                                o.raw_steering_angle= varargin{2};
                        elseif(nargin==2)
                                o.steering_angle= varargin{1}(1);
                                o.raw_steering_angle= varargin{1}(2);
                        end
                end
        end
end

% code to obtain the 10th steering data as an object
starting_time= importdata("starting_time_us.txt") ;
data = importdata("Bus_InterfaceCan_can0_Steering.txt"," ",0) ;
data_num_10=steering_data(starting_time,data(10,1:end)) ;
```

Text 3: Matlab code to import steering angle data

Wheel encoders

line	***ID_INTERFACE*_Wheels.txt**										
	Rtime	*dt_odo_right*	*speed_right*	*dist_right*	*dt_odo_left*	*speed_left*	*dist_left*	*raw_speed_right*	*raw_odo_right*	*raw_speed_left*	*raw_odo_left*

Details:

- *speed_right* is the speed of the right rear wheel, expressed in rad/s
- *dist_right* is the distance travelled by the right rear wheel, expressed in meters
- *speed_right* and *dist_right* are computed using the following formula:

$$speed\_right = kspeed\_right.rawspeed\_right$$
$$dist\_right = \frac{\pi.wheel\_diameter}{64}.rawodo\_right$$

  (128 is the number of tops by revolution)

  We suppose that kspeed as a resolution of 0.001 m/s and the diameter of the wheel (wheel_diameter) is 49cm (thus the resolution is approximately of 1cm).
- *speed_left* and *dist_left* are data for the left rear wheel.

With wheels about 40 cm in diameter, this represents approximately 2cm per "top" (0.40 * 3.14/64) - Specific calibration data (see [Calibration]) allow you to specify this value as we did above for the main odometer vehicle

Matlab

```
% Class
classdef odo_wheels_data
	properties (Access=public)
		dt_odo_right;
		speed_right;
		dist_right;
		dt_odo_left;
		speed_left;
		dist_left;
		rawspeed_right;
		rawodo_right;
		rawspeed_left;
		rawodo_left;
	end
	methods
		function o=odo_wheels_data(starting_time,varargin)
			if(nargin==10)
				o.dt_odo_right= varargin{1};
				o.speed_right= varargin{2};
				o.dist_right= varargin{3};
				o.dt_odo_left= varargin{4};
				o.speed_left= varargin{5};
				o.dist_left= varargin{6};
				o.rawspeed_right= varargin{7};
				o.rawodo_right= varargin{8};
				o.rawspeed_left= varargin{9};
				o.rawodo_left= varargin{10};
			elseif(nargin==2)
				o.dt_odo_right= varargin{1}(1);
				o.speed_right= varargin{1}(2);
				o.dist_right= varargin{1}(3);
				o.dt_odo_left= varargin{1}(4);
				o.speed_left= varargin{1}(5);
				o.dist_left= varargin{1}(6);
				o.rawspeed_right= varargin{1}(7);
				o.rawodo_right= varargin{1}(8);
				o.rawspeed_left= varargin{1}(9);
				o.rawodo_left= varargin{1}(10);
			end
		end
	end
end

% code to obtain the 10th steering data as an object
starting_time= importdata("starting_time_us.txt") ;
data = importdata("Bus_InterfaceCan_can0_Wheels.txt"," ",0) ;
data_num_10=odo_wheels_data(starting_time,data(10,1:end)) ;
```

Text 4: Matlab code to import odometry data (at the wheel level)

Accelerometers

line	*ID_INTERFACE*_Acc.txt								
	Rtime	accX	accY	accZ	acc1	acc2	acc3	raw_acc1	raw_acc2	raw_acc3

Details:

- accX, accY and accZ are the estimated values of the accelerations expressed in m/s^2 in the sensor frame
- acc1, acc2 and acc3 are the estimated values of the accelerations expressed in m/s^2 in each direction separatly
- *raw_acc1* , *raw_acc2* and *raw_acc3* are the raw values of the accelerations measured by the accelerometers (encoded in uint16_t; 65536 values). We supposed that the acceleration (accx for instance) is a linear function of the raw value:

$$\begin{cases} acc1 &= \frac{raw\_acc1}{kacc1} + offset\_acc1 \\ acc2 &= \frac{raw\_acc2}{kacc2} + offset\_acc2 \\ acc3 &= \frac{raw\_acc3}{kacc3} + offset\_acc3 \end{cases}$$

The values of the parameters according to the datasheet are: kacc1=kacc2=kacc3=1./(2*20.20125/65536.) and offset_acc1=offset_acc2=offset_acc3=0 (null offset and -20.20125 < acc (m/s2) < +20.20125). For our sensor, we estimate those values to be:

acc1	acc2	acc3
kacc1 = 1610.0	kacc2 = 1622.4	kacc3 = 1610.6
offset_acc1 = 0.935	offset_acc2 =-0.767	offset_acc3 = 1.876

The axis X,Y and Z are supposed to be aligned with the axis 1,2 and 3.

Matlab

```
% Class
classdef accelerometer_data
	properties (Access=public)
		accx;
		accy;
		accz;
		acc1;
		acc2;
		acc3;
		rawacc1;
		rawacc2;
		rawacc3;
	end
	methods
		function o=accelerometer_data(starting_time,varargin)
			if(nargin==2)
				o.accx= varargin{1};
				o.accy= varargin{2};
				o.accz= varargin{3};
				o.acc1= varargin{4};
				o.acc2= varargin{5};
				o.acc3= varargin{6};
				o.rawacc1= varargin{7};
				o.rawacc2= varargin{8};
				o.rawacc3= varargin{9};
			elseif(nargin==2)
				o.accx= varargin{1}(1);
				o.accy= varargin{1}(2);
				o.accz= varargin{1}(3);
				o.acc1= varargin{1}(4);
				o.acc2= varargin{1}(5);
				o.acc3= varargin{1}(6);
				o.rawacc1= varargin{1}(7);
				o.rawacc2= varargin{1}(8);
				o.rawacc3= varargin{1}(9);
			end
		end
	end
end

% code to obtain the 10th accelerometer data as an object
starting_time= importdata("starting_time_us.txt") ;
data = importdata("Bus_InterfaceCan_can0_Acc.txt"," ",0) ;
data_num_10=accelerometer_data(starting_time,data(10,1:end)) ;
```

Text 5: Matlab code to import accelerometer data

Gyrometer

line

ID_INTERFACE_Gyro.txt				
Rtime	wz	temp	raw_wz	raw_temp

Details:

- *wz* is the estimated value of the angular velocity, expressed in rad/s
- *temp* is the estimated value of the temperature, expressed in deg Celsus.
- *raw_wz* is the raw value of the angular velocity measured by the gyrometer (encoded in uint16_t). The estimated value is computed using the following formula:

  $$wz = \frac{1}{k\_wz} rawwz + offset\_wz$$

  According to the datasheet, -93.75deg/s < wz (deg/s) < 93.75deg/s and rawwz is encoded by 2 bytes (65536 values) thus we set k_wz=1./(187.5π/180./65536.) and offset_wz=-93.75π/180.
- *raw_temp* is the raw value of the temperature measured by the gyrometer (encoded in uint16_t). The estimated is computed using the following formula:

  $$temp = \frac{1}{k\_temp} rawtemp + offset\_temp$$

  The values of the parameters according to the datasheet are k_temp=1./(500./65536.) and offset_temp=-225.0 (null offset and -225 degC < temp (degC) < +275 degC)

Matlab

```
% Class
classdef gyroscope_data
	properties (Access=public)
		wz;
		temp;
		rawwz;
		rawtemp;
	end
	methods
		function o=gyroscope_data(starting_time,varargin)
			if(nargin==2)
				o.wz= varargin{1};
				o.temp= varargin{2};
				o.rawwz= varargin{3};
				o.rawtemp= varargin{4};
			elseif(nargin==2)
				o.wz= varargin{1}(1);
				o.temp= varargin{1}(2);
				o.rawwz= varargin{1}(3);
				o.rawtemp= varargin{1}(4);
			end
		end
	end
end

% code to obtain the 10th gyrometer data as an object
starting_time= importdata("starting_time_us.txt") ;
data = importdata("Bus_InterfaceCan_can0_Gyro.txt"," ",0) ;
data_num_10=gyroscope_data(starting_time,data(10,1:end)) ;
```

Text 6: Matlab code to import gyrometer data

Additionnal data

Dead-reckoning from odometry

The 2D positions and orientations of the vehicle are dead reckoned using knowledge about a vehicle's course and speed over the period of time using the motor and steering angle data (file _DeadReckoned_Poses.txt).

line ***ID_INTERFACE*****_DeadReckoned_Poses.txt**

Rtime	x	y	theta

Details:

- $[x;y]^T$ is the position of the vehicle estimated using odometry data, expressed in meters. When the application starts, $[x_{[0]};y_{[0]}]^T=[0;0]^T$.

- *theta* is the orientation of the vehicle estimated using odometry data, expressed in radians. When the application starts, $theta_{[0]}$=0.

$$\text{Motion time: } \Delta T = Atime\_measure_{[k]} - Atime\_measure_{[k-1]}$$
$$\text{Travelled distance: } \Delta S = \Delta T.transl\_vel$$
$$\begin{cases} \begin{bmatrix} x \\ y \end{bmatrix}_{[k]} = \begin{bmatrix} x \\ y \end{bmatrix}_{[k-1]} + \begin{bmatrix} \Delta S \cos(theta_{[k-1]}) \\ \Delta S \sin(theta_{[k-1]}) \end{bmatrix}_{[k]} \\ theta_{[k]} = theta_{[k-1]} + \Delta S \frac{\tan(steering\_angle)}{length\_rear\_axle} \end{cases}$$

(Ackerman's model)
We have supposed here that the steerting angle measurement time is the same as the translationnal velocity (which is not the case).
In our vehicle, the rear axle measures **length_rear_axle=1.21 m**.

Sample
Sample for interface of id *ID_INTERFACE*=Bus_InterfaceCan_can0 and vehicle sensor:

Bus_InterfaceCan_can0_DeadReckoned_Poses.txt

```
14816 0 0 0
34829 0 0 0
54829 0 0 0
74822 0 0 0
```

Dead-reckoning from odometry 2
A second file is provided where the 2D positions and orientations of the vehicle are dead reckoned using a geometric evolution model (file *ID_INTERFACE*_DeadReckoned_Poses2.txt). It can be more accurate than the first file because it does not use the velocity (which is computed by integrating the motor encoder value by the time) but directly the displacement.

line ***ID_INTERFACE*_DeadReckoned_Poses2.txt**

Rtime	x	y	theta

Sample
Sample for interface of id *ID_INTERFACE*=Bus_InterfaceCan_can0 and vehicle sensor:

Bus_InterfaceCan_can0_DeadReckoned_Poses2.txt

```
14816 0 0 0
34829 0 0 0
54829 0 0 0
74822 0 0 0
```