Vehicle Counting – Application Note

Overview

Vehicle counting is an application that is totally solved with uRAD Industrial model + Tracking Software. It allows you to count vehicles in multiple lanes, measuring the velocity and classifying them, with high accuracy and minimal configuration. The system works at the 60 GHz band, an available frequency band for emission all over the world, which ease the certification of any product.

Use Case

The system is very versatile and can be used in many counting scenarios:

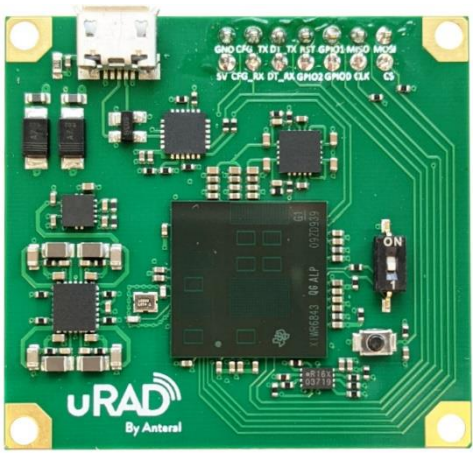
- Urban or interurban roads.
- Velocity measurement up to 180 km/h.
- Up to 6 lanes monitoring with a single radar.
- Counting vehicles with positive velocity (moving away) and negative velocity (approaching) at the same time.
- Dense or light traffic scenarios

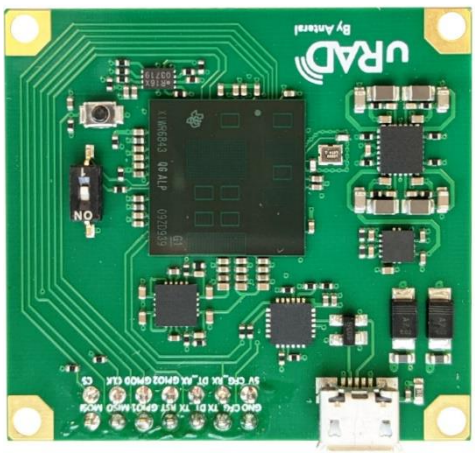
Mounting

Three aspects have to be taken into account when mounting your vehicle counting system:

- **Radar orientation**

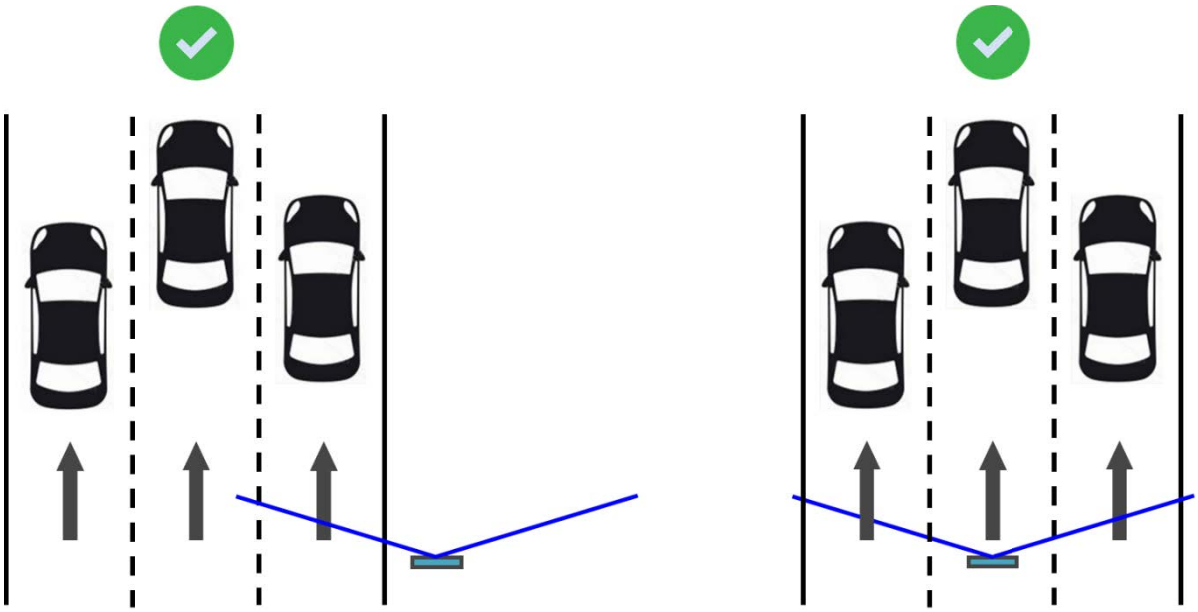
uRAD Industrial has to be mounted in the way that the USB connector is in the upper or lower side. Both options are correct.



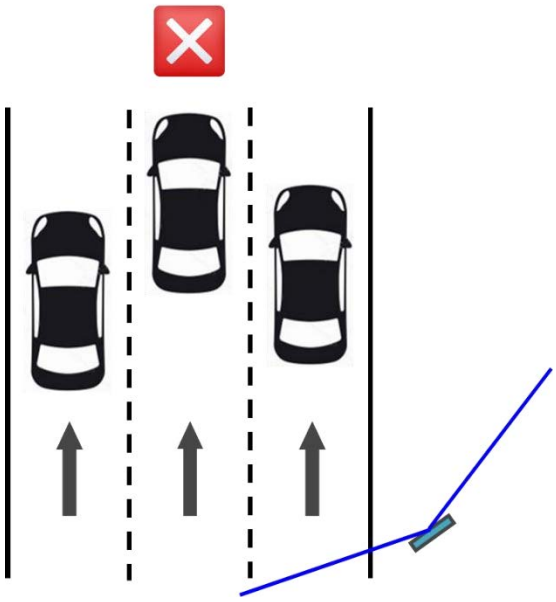


- **Radar mounting in respect of the road – yaw angle**

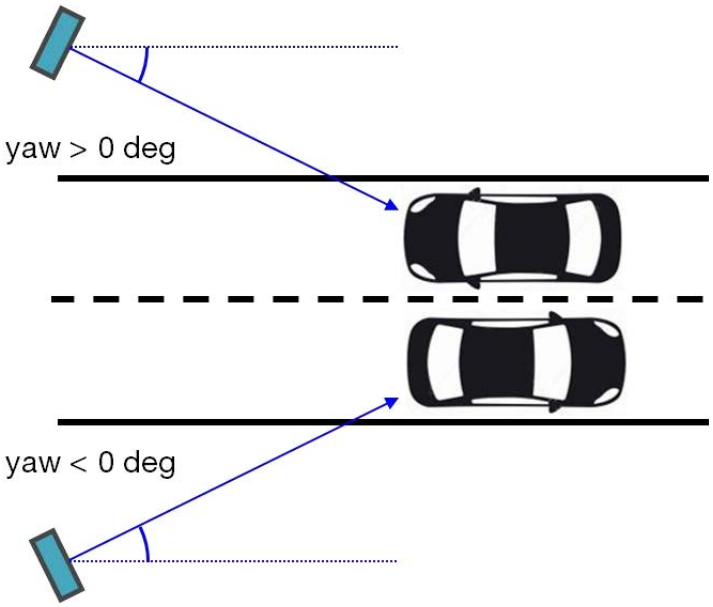
uRAD Industrial can be mounted at one side of the road or above it. **The most important thing is to mount it parallel to the road**. See the following images.



It is not necessary to turn the radar to point to the center of the road. The accuracy will be compromised. Yaw angle has to be 0 degrees. The field of view is enough to measuring several lanes with parallel mounting.



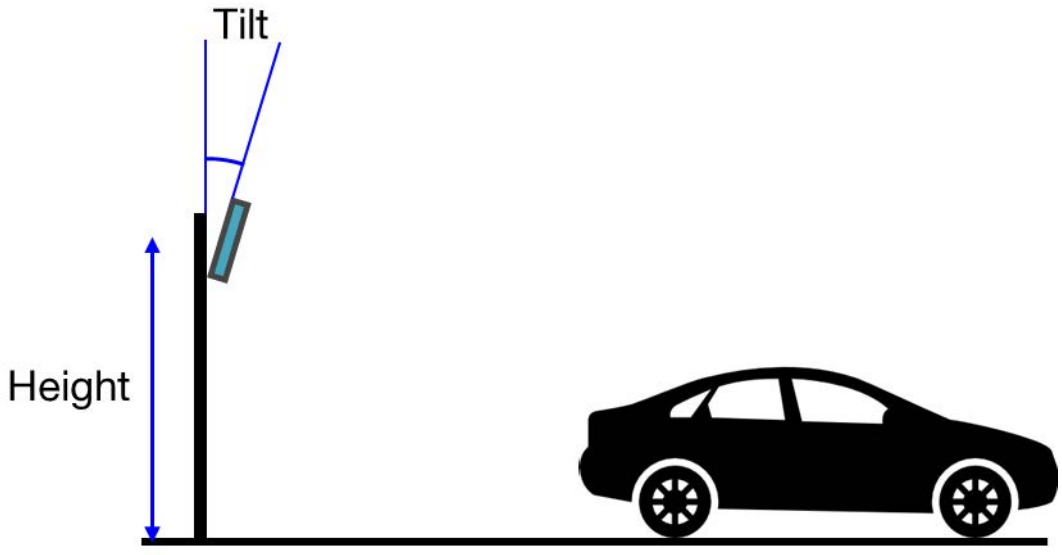
In the case it is **absolutely necessary** to give some yaw angle because the radar is too far from the nearest lane or some object is partially blocking the field of view, take into account the sign of the angle for the configuration parameters according to the next image.



- **Radar height and tilt**

In order to have proper view of the vehicles and thus, avoid blocking from one vehicle to another, the radar has to be placed at a height of 3 meters or above. Depending on the mounting height, the radar has to be tilted downward, but only a bit. Follow these recommendations.

HEIGHT	TILT
2 m to 3 m	0 degrees
3 m to 4 m	5 degrees
4 m to 5 m	10 degrees
5 m to 6 m	15 degrees



- **View distance and detection mode**

The optimal detection distance of vehicles is 20 meters. That is why it is advisable to install the radar on roads where traffic can be monitored, **without it stopping**, between a distance of 10 to 40 meters from the radar.

Therefore, avoid placing it near traffic lights or intersections where traffic may stop and accumulate.

It is not always possible to place it in a place where traffic is fluid between 10 and 40 meters, therefore, there are two modes of operation:

- **Standard mode**: there is enough length to monitor traffic without stopping between 10 and 40 meters.

- **Fast mode**: only a distance between 5 and 20 meters is available to monitor traffic without stopping.

It is advisable to use the standard mode since by monitoring the vehicles over a longer distance, the count is more accurate where the traffic is very dense, as well as the speed measurement.

In addition, it is **necessary to install the device on a straight stretch of road** along the detection distance. **Avoid installation in sections with curves.**

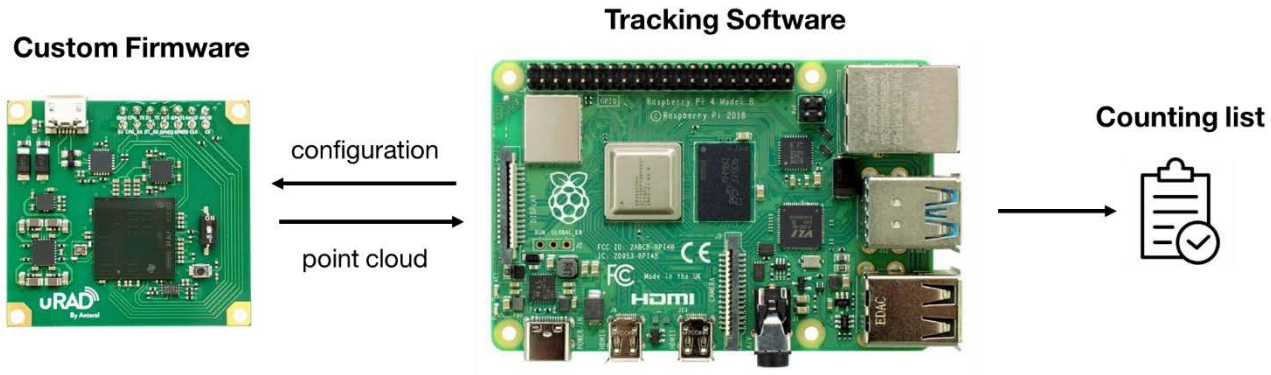
Tracking Software

The Tracking Software consists of a Python program that has to be run in the master device that controls uRAD Industrial. This software together with the master devices:

- Sends to uRAD Industrial the corresponding configuration parameters.
- Receive from uRAD Industrial the 3D point cloud with X, Y, Z space coordinates, velocity and SNR (Signal to Noise Ratio).
- Process the point cloud to identify vehicles, extract their velocity and classify them.
- Save a counting list with the relevant information.

The type of target identifies, in first instance, pedestrians, regular vehicles and large vehicles (big trucks). Moreover, the classification can be refined to include bicycles.

Any device where a python program run and that has UART/USB communication can be used as the master device. A typical example of a device that can be easily integrated with uRAD Industrial is a Raspberry Pi.



Tracking software is not included by default with the purchase of uRAD Industrial. Must be purchased separately. In addition, uRAD Industrial carries a custom Firmware that cannot be modified, so that unit will only work for this application. Contact us for more purchasing information.

• Python files

The Tracking Software is comprised of a main Python script and a compiled library.

The main script, named ***vehicleCounter_smartcities.py***, is open source and it **is the one that has to be run**. The library, named ***uRAD_Tracking***, is compiled and it is where the point cloud data is processed.

The Python version for running the software on Raspberry Pi is 3.9 and on Windows it can be used the version 3.9 or 3.10.

We provide a single script, that can be used with UART communication, assuming, for example, that the radar is used with the PCB adapter + Raspberry Pi, or with USB communication, for instance with a PC. Some configuration parameters have to be set depending on the use case.

- **Communication interface**

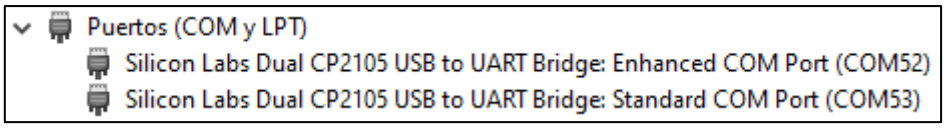
Select True or False if the radar is connected by USB or by UART, respectively.

```
#### COMMUNICATION INTERFACE ####
usbCommunication = True
```

Some lines below, set the port names of USB or UART communication.

```
# Ports names
if (usbCommunication):
    # Specify serial port names (Enhanced and Standard)
    configPort_name = 'COM52'
    dataPort_name = 'COM53'
    # When connected to PC you have not access to reset pin, unless you configure
    reset = False
else:
    # Name of serial port (/dev/serial0 in Raspberry Pi)
    Port_name = '/dev/serial0'
    reset = True
```

When uRAD Industrial is connected by USB, two COM ports are recognized. **configPort** is the one identifies as **Enhanced** and **dataPort** is the **Standard.**



If connected by UART and therefore, `usbCommunication = False`, by default, the `port_name` is the one for Raspberry Pi (`/dev/serial0`). It is necessary to enable the serial port in *Preferences > Raspberry Pi Configuration > Interfaces* and activate *Serial Port: Enable*, since it is possible that is not enabled by default.

Moreover, it is also assumed that uRAD is used together with the uRAD PCB adaptor for Rasbperry Pi, that connects the uRAD Reset pin with GPIO number 6 of Raspberry Pi. If the reset pin is not connected or connected to other RPi pin, modify the lines accordingly.

- **Configuration parameters**

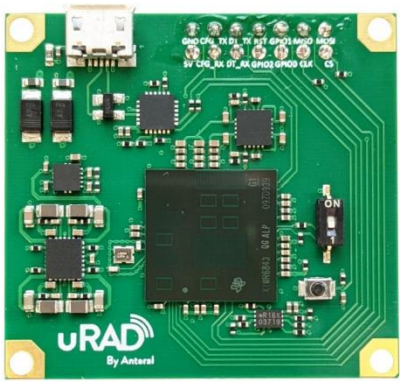
Just a few input configuration parameters have to be set in ***vehicleCounter_smartcities.py***. At the beginning of the script, there are some lines to introduce the configuration variables:

```
#### CONFIGURATION PARAMETERS ####
usbConnector_upward = False                 # Define the radar orientation by
pitch_angle = 15                            # Tilt angle with the vertical in the
yaw_angle = 0                               # Yaw angle relative to the road in
velocityPositive = True                     # Counting vehicles driving away from
velocityNegative = True                     # Counting vehicles approaching to
frequencyChannel = 3                        # Frequency channel within the 60-64
```

```
x_min = -20                          # Minimum distance to measure in the
x_max = 20                           # Maximum distance to measure in the
tracking_mode = "standard"           # Tracking vehicle range (option
```

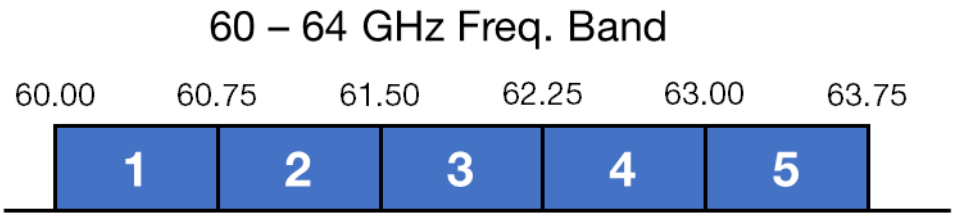
- **usbConnector_upward:** define the radar orientation by means of the position of the USB connector. Set this variable according the indications in the following image.

usbConnector_upward = True



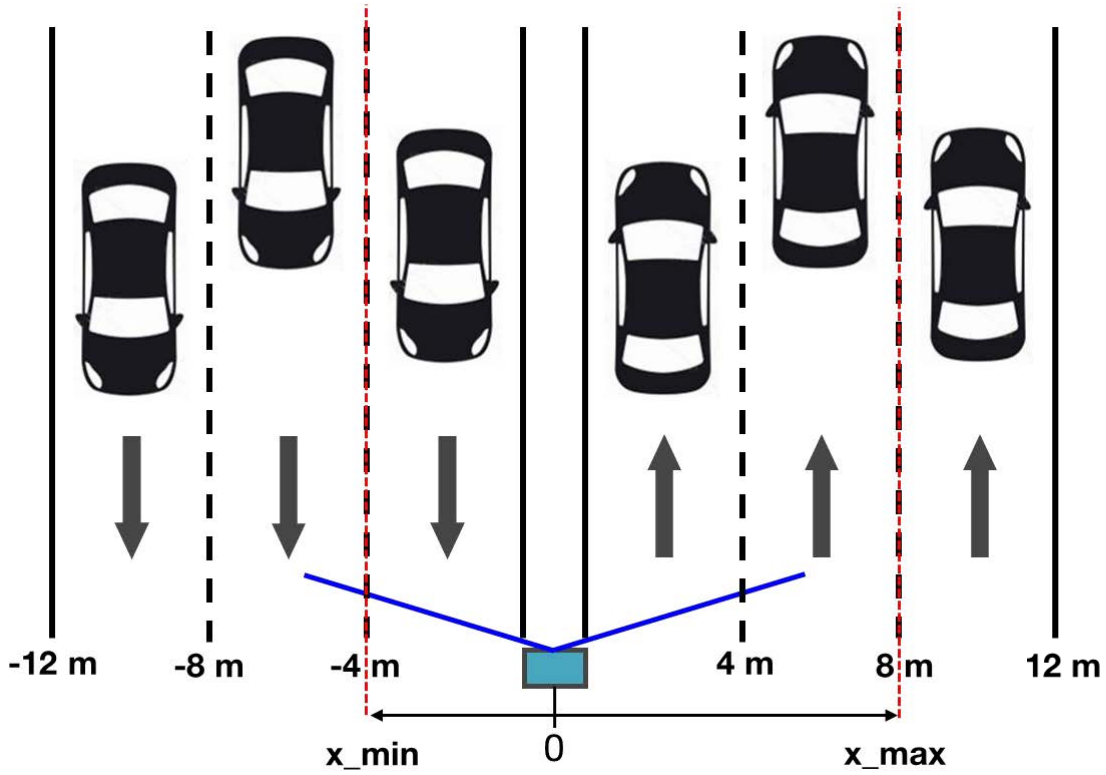
usbConnector_upward = False

- **pitch_angle**: define the tilt angle of the radar with the vertical (in degrees). Set this parameter according to your mounting.
- **yaw_angle:** define the yaw angle of the radar in respect to the road (in degrees). Set this parameter according to your mounting. **Pay attention to the sign of yaw angle depending on your particular installation at left or right to the road**.
- **velocityPositive**: set True/False for counting/discarding vehicles driving away from the radar.
- **velocityNegative**: set True/False for counting/discarding vehicles approaching the radar.
- **frequencyChannel**: uRAD Industrial can work in five different channels in the whole 60-64 GHz frequency band. This is useful, for example, for fulfilling with specific region regulations or for limiting interference between near devices. Set this parameter from 1 to 5 according the following image.



- **x_min**: defines the minimum distance in meters in the horizontal direction that you want to consider to count vehicles. This is useful for limiting the number of lanes to monitor.
- **x_max**: defines the maximum distance in meters in the horizontal direction that you want to consider to count vehicles. This is useful for limiting the number of lanes to monitor.

Imagine, with the following example, that you want to measure only the two lanes at the right of the radar and one lane at the left of the radar. You have to set x_min = -4 and x_max = 8.



- **tracking_mode**: allows you to set the distance on the longitudinal axis (**y**) at which vehicle tracking is created. One of these modes must be selected:
  - **standard**: Vehicle tracking is performed between y=7.5m and y=40m. This is the recommended mode for most applications.
  - **fast**: Vehicle tracking is performed between y=5m and y=20m. This configuration is recommended only in applications where it is necessary to **obtain the notification of the new vehicle as soon as possible or the radar installation does not allow the detection of vehicles up to 40 meters.**

- **Output results**

User can select to save three types of results:

```
#### OUTPUT RESULTS ####
saveResults = True                          # Save counting results
saveVehicleFeatures = True                  # Save vehicle features
saveRawData = True                         # Save the full point cloud
outputDatetimeFormat = 0                    # 0 for datetime (yyyy/mm/dd HH:MM:SS
```

- **saveResults:** creates a .txt named **Vehicle_results.txt** with the most relevant information. Each line of this text file corresponds with one detected vehicle. The information of each column is the following:

  `Timestamp Velocity x_distance Vehicle_type`

  `Timestamp`: date and time (yyyy/mm/dd HH:MM:SS) or timestamp (UNIX) the vehicle is detected. It is taken from the device system, the Raspberry Pi in this case.

  `Velocity`: velocity in km/h of the vehicle. Positive velocity means vehicle driving away and negative velocity means vehicle approaching.

  `x_distance`: horizontal distance estimation in meters of the vehicle. Useful for lane identification.

  `Vehicle_type`: vehicle type identification. 1 = regular vehicle, 2 = long vehicle, 3 = pedestrian. For a correct classification, the classifier algorithm must be configured to each particular scenario.

  A long vehicle is any vehicle detected with a length of more than 15 meters.

  A pedestrian is any vehicle detected with a speed of less than 10 km/h.

- **saveVehicleFeatures:** creates a .txt named **Vehicle_features.txt** with additional information for each vehicle. **This information is useful for uRAD team to check the vehicle classification and to improve it for a particular scenario.**

  Each line of this text file corresponds with one detected vehicle. The information of each column is the following:

  `Timestamp Velocity x_distance x_distance_std Density Density_long Length_long`

  `Timestamp`: date and time (yyyy/mm/dd HH:MM:SS) or timestamp (UNIX) the vehicle is detected. It is taken from the device system, the Raspberry Pi in this case.

  `Velocity`: velocity in km/h of the vehicle. Positive velocity means vehicle driving away and negative velocity means vehicle approaching.

`x_distance`: horizontal distance estimation in meters of the vehicle. Useful for lane identification.

`x_distance_std`: standard deviation of the horizontal distance estimation.

`Density`: point density associated to the vehicle.

`Density_long`: when a vehicle is susceptible to be a long vehicle, the point density associated at short distances. 0 means that this vehicle is directly identified as a regular vehicle.

`Length_long`: when a vehicle is susceptible to be a long vehicle, indicates an estimation of its length. 0 means that this vehicle is directly identified as a regular vehicle. Do not take this number as a true value.

- **saveRawData:** creates a .txt named **PointCloud_stats.txt**. This information is useful for uRAD team to check the proper working of the radar.

  This file contains the full 3D point cloud. Each line starts with the radar frame number and its corresponding time stamp. Subsequently each line contains (X,Y,Z,velocity,SNR,noise) of all points in that raster.

- **outputDatetimeFormat:** to choose the format of the **`Timestamp`** field in .txt files. 0 for date and time (yyyy/mm/dd HH:MM:SS), 1 for timestamp (UNIX).

The path and name of the files can be chosen in the code a few lines below:

```
foldername = 'Results'                          # Folder where output files will be
output_filename = 'Vehicle_results.txt'         # Name of counting results file
features_filename = 'Vehicle_features.txt'  # Name of vehicle features file (for
pointCloud_filename = 'PointCloud_stats.txt'      # Name of radar pointCloud file
```

- **Offline Analysis**

Besides the scripts for real time measuring, an additional script for offline analysis is included. This script, named **plot_results.py**, reads the file **PointCloud_stats.txt** and simulates what the tracking software does in real time.

At the configuration parameters section, you have to introduce the same configuration that you set when recorded the .txt.

```
#### CONFIGURATION PARAMETERS ####
test = 1                                    # Folder number where results are saved
usbConnector_upward = False                 # Define the radar orientation by means
pitch_angle = 15                            # Tilt angle with the vertical in the
yaw_angle = 0                               # Yaw angle relative to the road in the
velocityPositive = True                     # Counting vehicles driving away from
velocityNegative = True                     # Counting vehicles approaching to the
frequencyChannel = 3                        # Frequency channel within the 60-64
x_min = -20                                 # Minimum distance to measure in the
x_max = 20                                  # Maximum distance to measure in the
tracking_mode = "standard"                  # Vehicle tracking range on
```

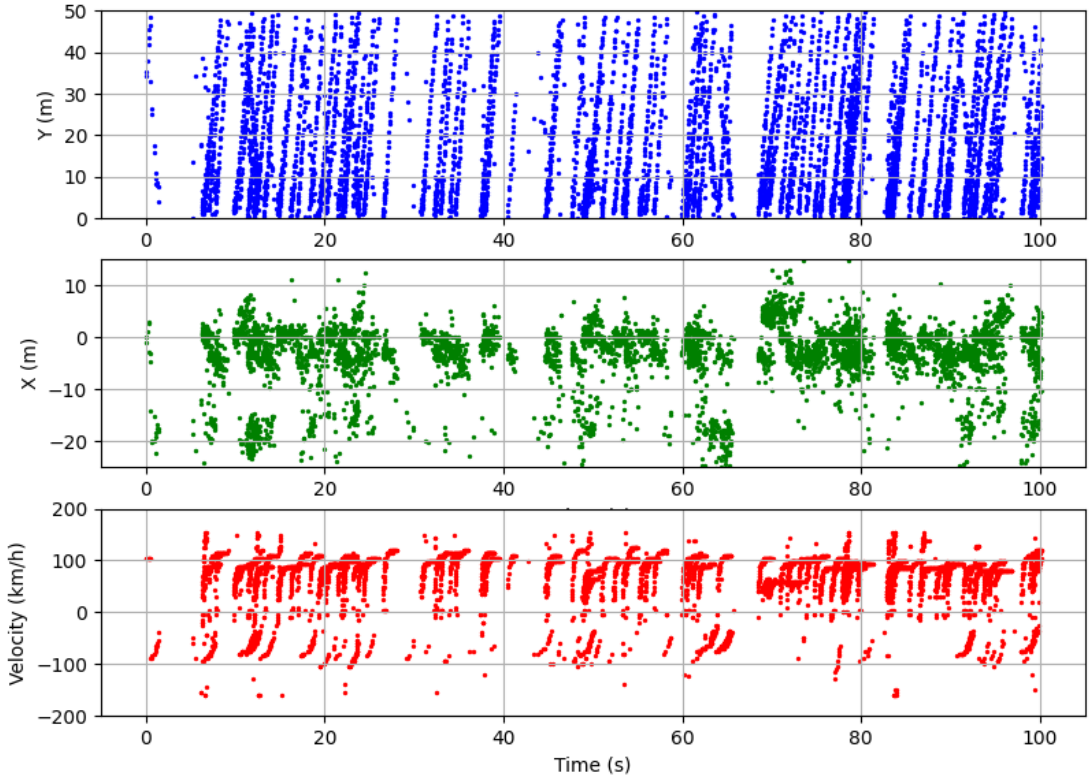
You can select to generate the results files **Vehicle_results_offline.txt** and **Vehicle_features_offline.txt** like if they were generated in real time.

```
#### OUTPUT RESULTS ####
saveResults = False                         # Save counting results
saveVehicleFeatures = False                 # Save vehicle features
output_filename = './Test/%d/Vehicle_results_offline.txt' % test
features_filename = './Test/%d/Vehicle_features_offline.txt' % test
outputDatetimeFormat = 0                    # 0 for datetime (yyyy/mm/dd HH:MM:SS
```

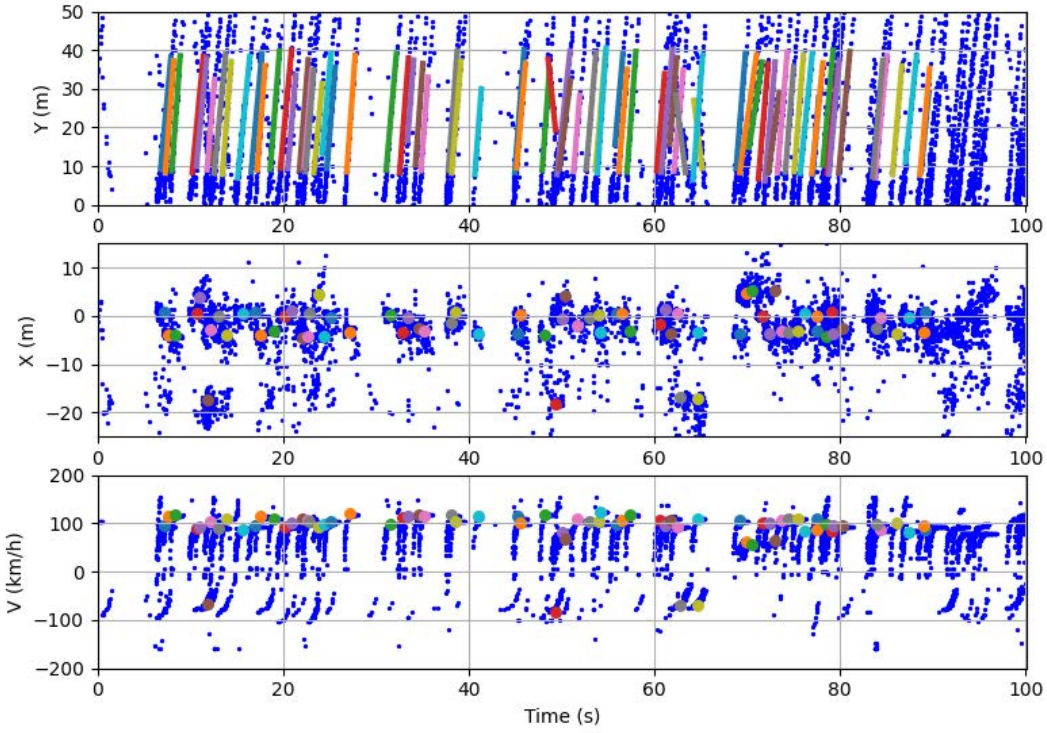
And two different graphs:

```
plotPointCloud = True                       # Plot Distance, X and Velocity from
doTracking = True                           # Do the tracking simulation
plotTracking = True                         # Plot X,Y,Z from PointCloud.txt and
tracking lines over the point cloud
plotTrackingWithPointCloud = True           # Plot both previous plots together
plotTrackingMode = 0                        # Tracking plot mode (0 or 1). 0 =
```

- **plotPointCloud**: represents **y** points (point in the longitudinal direction), **x** points (points in the horizontal direction) and **velocity** points.



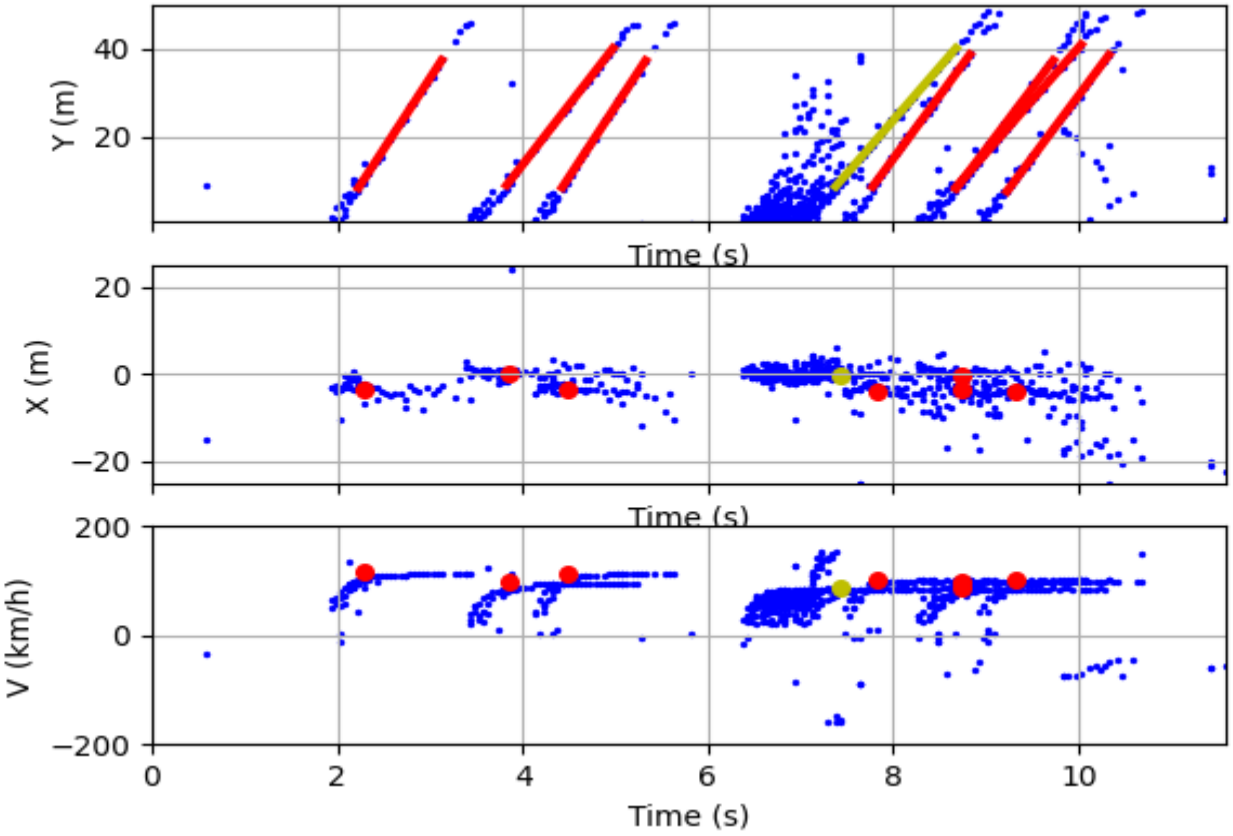
- **plotTracking and plotTrackingWithPointCloud**: represents **y, x** and **velocity** points, as in the previous plot but the tracking of each vehicle is also plotted as color lines or color dots.



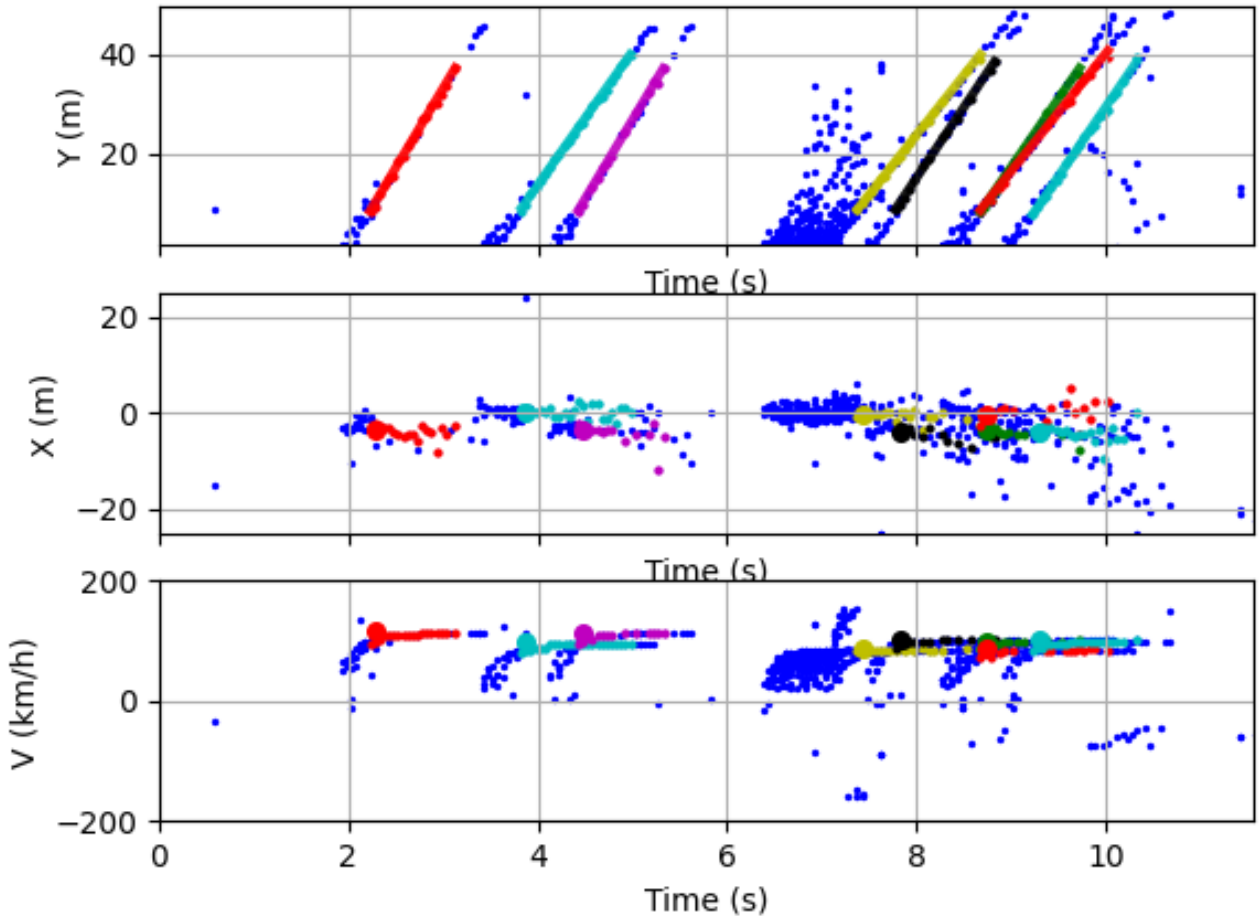
- **plotTrackingMode**: Allows you to configure the above plot by coloring the vehicles in different ways. The available modes are:

0: Each vehicle is plotted with the color corresponding to its type.

Type	Color
Regular vehicle (type 1)	Red
Long vehicle (type 2)	Yellow
Pedestrian (type 3)	Green



1: Each vehicle is plotted in a different color, coloring both the line representing the vehicle and the points associated with it, with the same color.



Moreover, the console prints one line for each vehicle with the time, velocity, horizontal distance and type.

```
New vehicle, time: 7.21 s, velocity: 94.76 km/h, x: 0.65 m, type: 1
New vehicle, time: 7.66 s, velocity: 115.34 km/h, x: -4.00 m, type: 1
New vehicle, time: 8.34 s, velocity: 119.30 km/h, x: -4.07 m, type: 1
New vehicle, time: 10.66 s, velocity: 90.18 km/h, x: 0.64 m, type: 1
...
```

Graphical User Interface

Moreover, a Graphical User Interface (GUI) is provided to run the tracking software in a more visual and intuitive way.

For Windows, the file *vehicleCounter_smartcities_GUI.exe* has to be executed. In the Raspberry Pi, the python program *vehicleCounter_smartcities_GUI.py* has to be run.