



VehicleCounter 3.3

**User manual
for v3.3.80+**

Introduction

Hello! Thank you for choosing Floud's VehicleCounter!

VehicleCounter is a traffic monitoring application developed by Floud srl (<http://www.floud.eu>) on the Axis Camera Application Platform (ACAP, <http://goo.gl/uGesw>) and as such it can be installed on most of the ACAP-enabled Axis devices. The application is capable of collecting meaningful statistics regarding the number of vehicles passing along the observed lanes, and their estimated speed and length. Based on these information, vehicles are assigned to one out of six classes.

VehicleCounter 3.0+ (VC3 in short) uses *virtual sensors* that can be drawn directly on the images grabbed by the camera after its installation on-site. Sensors configuration can be made by means of a web application which is installed on-board and served by the camera web server. For each detected vehicle, a record in a table of an on-board SQLite3 database is written. Collected data are stored on-board in the internal memory or SD card, and made available through a set of simple, yet complete, CGI calls. Stored data are retained for a configurable number of days and then canceled to prevent storage space shortage.

How to read this manual

VC3 is a complex application. It uses computer vision techniques to detect events occurring in the real world through images of it collected by a video sensor. Computer vision applications are usually hard to set up correctly because there are a lot of parameters to tune, a lot of variables to take into account, and a lot of problems to deal with. We have spent a lot of efforts in the attempt of making VC3 as easy to set up and configure as possible. Nevertheless, there is still a lot to know about if you want to be able to get the maximum performance out of VC3. That said, we organized this user manual as follows.

Part I is dedicated to preliminary operations that you need to carry out to prepare things before you can even be able to start VC3. This operations include getting VC3 package and license key, setting up an ACAP-enabled camera properly, installing VC3 on the camera etcetera.

Part II is a quick start guide that explains the basic features of VC3 and how to get it to work properly. This includes setting up a new configuration, a little parameters tuning, fixing most common problems and how to read data from VC3. This reading should be enough to start collecting meaningful data.

Part III is a comprehensive guide to VC3. If something has been left out for briefness or clarity before, now we will tell you the whole story in detail. Read this when you really want to go deep into VC3 features and try to draw every single drop out of VC3.

Features

VC3 offers the following features:

- accurate counting of vehicles for each lane and direction
- length and speed estimation for each vehicle

- classification of each vehicle (based on estimated length) in six classes: unknown vehicles, motorcycles and small cars, medium cars, large cars and vans, trucks, large trucks
- per-vehicle or aggregate statistics
- detection of particular events of interest, such as heavy traffic, queue formation, wrong way vehicle
- powerful web services API
- integration with Axis events system
- JSON¹ data format
- easy configuration through web interface

Requirements

As an embedded application for the Axis Camera Application Platform, an Axis camera server (or video server) is needed in order to run VC3. The user is in charge of buying a camera that is compatible with VC3. Read further on in this manual for more info about compatibility with Axis cameras, or contact Floud srl for information on a specific camera model.

To be able to configure VC3 and to query the data collected by VC3 the user needs a device capable of sending customized HTTP requests to the host where VC3 is installed, i.e. a common PC (any O.S. will work) or a mobile device (smartphone or tablet), connected to the Axis camera via wired or wireless Ethernet connection. The most common web browsers can be used to access our configuration pages. Google Chrome and Mozilla Firefox have been tested and are recommended. Other browsers may work just fine, however older versions of Microsoft Internet Explorer are known

to be incompatible with VC3 and are NOT supported.

Disclaimer

The VehicleCounter software is provided "as is" without warranty of any kind. By using it, the user accept to relieve Floud srl from any responsibility for any event that could possibly happen as a consequence of the use of VehicleCounter. Thus, Floud srl will not be responsible for damages possibly occurring to connected software or hardware, nor of any data loss or corruption. Also, Floud srl will not be responsible for any damage possibly occurring to persons and/or things as a consequence of decision taken by anybody with the aid of the information produced by VehicleCounter. The information that VehicleCounter produces must be considered as rough estimates of the figures that they refer to. Accuracy of these estimates can vary according to a wide set of unconstrained parameters upon which nor VehicleCounter neither Floud have any degree of control. Floud does not give any guarantee upon the level of performance achievable by VehicleCounter in any situation.

¹Javascript Object Notation: <http://goo.gl/6gMz>

Copyright notice

All the material included in this manual is ©2017 Floud srl. All other company names and products are trademarks or registered trademarks of their respective companies. We reserve the right to introduce modifications without notice.

Contents

I VehicleCounter 3 for Axis Camera Application Platform	6
1 Installation	6
2 Licensing	6
3 Camera setup	8
4 Running the application	9
II Getting started	10
5 Operating principle	10
6 Setting up a simple configuration	11
7 Diagnostics	12
8 Fixing common configuration problems	13
8.1 Fixing vehicles detection and counting	13
8.1.1 Vehicle missed	14
8.1.2 False positives (double counting)	15
8.2 Fixing length and speed estimation	15
9 How to get the data	16
9.1 On-demand data transfer	16
9.2 Automatic data transfer	17
10 Data charts	18
III Comprehensive guide	19
11 Application parameters	19
12 VehicleCounter configuration page	21
12.1 Configuration tree view	22
12.2 Real time data	24
12.3 Live preview	25
13 Configuration guidelines	26
13.1 General principles	26
13.2 Drawing the counter sensor	26
13.3 Counter sensor's parameters	27
13.4 Drawing the gates	28

13.5 Gate sensor's parameters	29
14 Triggering events	30
15 Web API	31
getDiagnostics.cgi	32
getProbeState.cgi	33
getCount.cgi	34
getRawData.cgi	34
getConfiguration.cgi	36
setConfiguration.cgi	38
getPassages.cgi	39
getStatsPerClass.cgi	40
getSpeedDistribution.cgi	42
getLengthDistribution.cgi	43
getBuildVersion.cgi	45
cleanDB.cgi	45

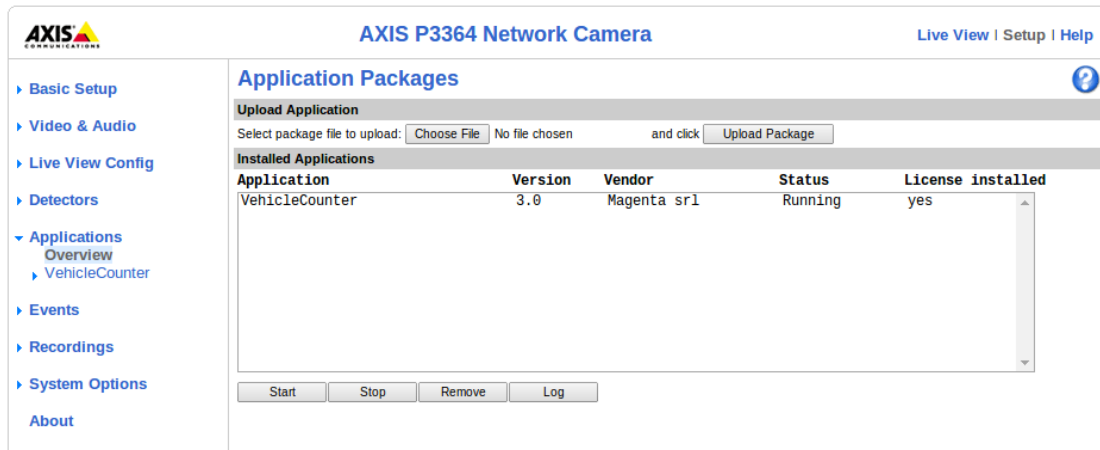


Figure 1: The 'Application Packages' page with VC3 installed, up and running.

Part I

VehicleCounter 3 for Axis Camera Application Platform

1 Installation

VC3 is compatible with most of the ACAP-enabled Axis camera servers. Particularly, all camera models running Axis firmware of version 5.80.* or above, should be able to run VC3. Older firmware versions may require older versions of VehicleCounter as well. These versions may be available but are not actively developed. Write an email to info@floud.eu if you want more information on a specific camera model, or firmware version, or legacy version of VehicleCounter.

To install VC3 on an ACAP-enabled Axis camera, it is first needed to obtain the application EAP package provided by Floud srl. If you do not have one, you can request it by writing an email to info@floud.eu. Then, follow these steps:

1. access the camera server web interface and browse to the "Application Packages" page.
2. in the section "Upload Application" choose the VC3 EAP package and click the "Upload Package" button.
3. once installed, the VehicleCounter application will be listed in the "Installed Application" section of the page, as depicted in Fig. 1 (VehicleCounter version may vary).

2 Licensing

In order to be able to run VehicleCounter, the user must install a valid license key file. To obtain it, the user must visit the Axis website at <http://www.axis.com/global/en/products/>

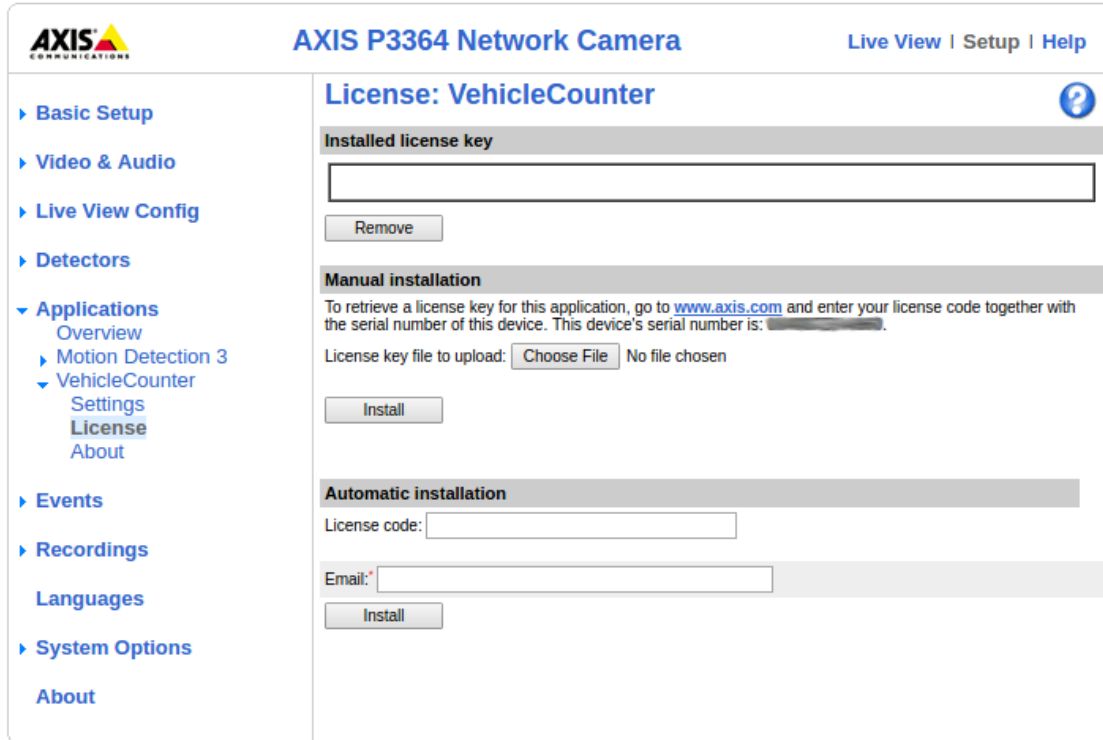


Figure 2: The VehicleCounter 'License' page.

camera-applications/license-key-registration (registration on Axis website required).

For a 30-days trial license key, proceed as follows:

1. enter your camera serial number;
2. select "I'd like to create a trial or a free license" radio button
3. type VehicleCounter in the text box (suggestions will appear as you type)
4. click "Generate"

A license.key file will be generated and made available for download through a link on the web page. Download it and store it to a safe place. Please note that the 30-days trial starts when you generate the license.key file, NOT when you install VC3 or the license file itself.

For a full license key, you first need a license code, and that will be provided by Floud srl when you purchase VC3. After you received the license code, visit <http://www.axis.com/global/en/products/camera-applications/license-key-registration> and proceed as follows:

1. enter your camera serial number;
2. select "I have a license code" radio button
3. enter the license code in the text box
4. click "Generate"

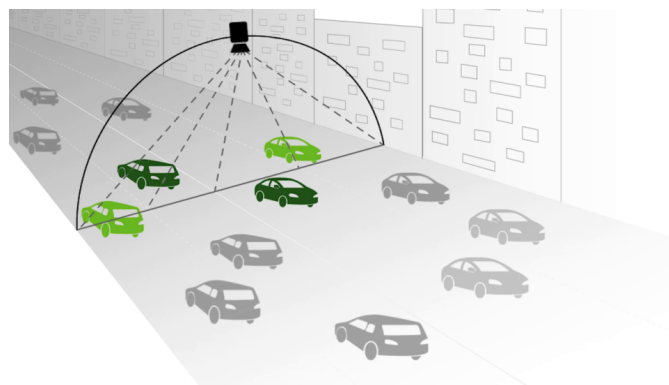


Figure 3: The *zenithal* point of view is optimal, as it minimizes perspective distortions.

A license.key file will be generated and made available for download through a link on the web page. Download it and store it to a safe place.

The license key can be installed by accessing to the VehicleCounter “License” page (see Fig. 2). The camera interface provides the following installation methods for license keys:

1. **Manual install.** If a license key file is already available, the user can upload it manually to the camera server via the “License” page of the Vehicle Counter application (“Choose file...” and “Install” in the Manual Installation).
2. **Automatic install.** If only a license code is available, it can be used in the “Automatic Installation” section in order to receive the license key file directly from Axis. To do this, the user must insert the license code and a valid e-mail address in the “Automatic installation” section of the “License” page of the Vehicle Counter application. By clicking “Install”, these informations, together with the camera serial number will be transmitted to Axis, which will respond with a valid license key that will be automatically installed. This method obviously requires that the camera is connected to the internet and is correctly configured.

Once a valid license key has been installed, the application can be started (see Sec. 4).

3 Camera setup

Traffic monitoring applications are typically designed to work from an elevated point of view, in order to reduce perspective in the observed scene. Unfortunately, in real scenarios this is rarely achievable. That is why Floud’s VC3 is designed to work also in situations where a certain amount of perspective is unavoidable.

Ideally, the best camera setup is the one that guarantees a zenithal view of the scene (see Fig. 3). This ensures the lowest occurrence of occlusions and lens distortions and the maximum accuracy in car detection and counting. However, this setup is often not achievable, and the camera has to be placed in sub-optimal positions. VC3 can work efficiently nevertheless, provided that the actual setup gives limited perspective deformation. Performance will progressively degrade as long as camera elevation reduces and perspective increases (see Fig. 4).

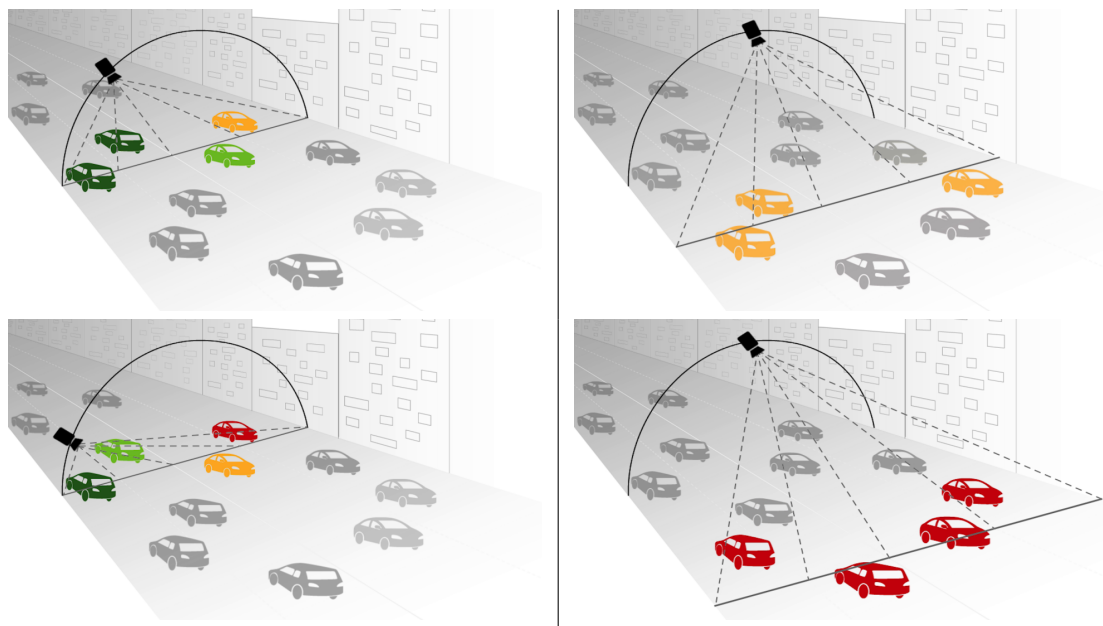


Figure 4: Effect of occlusions and perspective distortion for different points of view: performance will progressively degrade as perspective increases.

In our test setup, the camera is mounted at approximately 9 meters of height, and observes a 4-lane road with an angle of about 45° . In this setup, vehicle count proved to have an accuracy above 90% with different lighting and weather conditions.

4 Running the application

To run the application, select it from the list in the “Application Packages” page of the camera server web interface and click the “Start” button (see Figure 1). The application status should change from “Not Running” to “Running”. To stop the application simply click the “Stop” button.

In case of power loss, Vehicle Counter will be restarted automatically, if it was running when the power failure occurred.

The VC3 application can be setup entirely using the camera web interface. Once the application is successfully installed, a “VehicleCounter” menu will appear under the “Application” menu in the left column of the Axis camera web page. By selecting it, the user can access to the “Settings”, “License” and “About” pages. From the “Settings” page, the link “Main page” leads to the VC3 Configuration Page, which is the entry point for all VC3 configuration and diagnostics tools.

Part II

Getting started

5 Operating principle

VC3 allows the user to define *probes* that are the sensitive elements through which the application extracts information from the images collected by the video sensor. The simplest probe is made of a single virtual sensor called *counter sensor* (see Fig. 5). The counter sensor is just a poly-line that the user places in the desired position of the image and is capable of counting objects that moves through it. Such a probe is called *free probe*, because the counter sensor is always enabled and counts whatever object crosses it regardless from the direction of motion. A slightly more complex probe uses a different kind of sensor, called *gate*, together with the counter sensor. A gate is an oriented poly-line (most often it will be a single rectilinear segment, but it can be more complex than this) that is sensitive to apparent motion occurring in a given direction (called *positive direction* of that gate). A probe that has at least a gate, beside its counter sensor, is called *gated probe* (see Fig. 6). Unlike free probes, gated probes have their counter sensor normally disabled, unless at least one of its gates detects persistent motion in its positive direction. When this happens, that gate enables the counter sensor so that it can count the moving object. If none of the probe's gates detects motion, the counter sensor is disabled again.



Figure 5: Examples of free probes with different geometries in different perspective conditions.

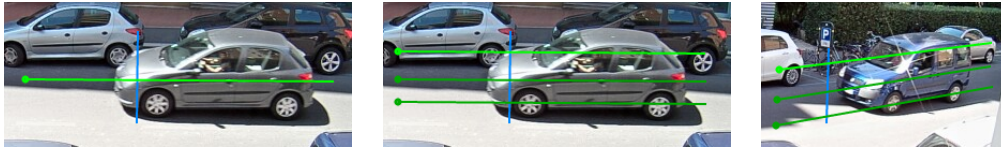


Figure 6: Examples of gated probes with different geometries in different perspective conditions.

This mechanism is the basics of VC3 operations. By placing custom-shaped gated probes over the image, the user can selectively count vehicles crossing relevant section of the image in the desired direction. This feature allows VC3 to run effectively even from low points of view (see Fig. 4), where the perspective would usually make counting infeasible due to occlusions between vehicles moving in opposite directions. With VC3, instead, this situation is tolerated because the correct positioning of gates in a probe allows to filter out everything that moves in the wrong direction. Also, detection of “wrong way vehicles” is signaled by triggering an alarm that can be catch by the Axis camera’s event system.

6 Setting up a simple configuration

Fig. 7 shows VC3 configuration page. It can be accessed through the link “Main page” in the “Application Settings: VehicleCounter” page on your Axis camera. To set up a simple configuration, access the configuration page with your web browser and follow these steps.

1. If needed, delete the current configuration by clicking on the “X” icon of each probe in the configuration’s tree view.
2. Click the “Add probe” command. A new probe is added to the configuration’s tree view and the newly created probe is selected (the probe’s name is surrounded with a dashed rectangle).
3. Draw the counter sensor by left-clicking on the video preview pane. Each click adds a corner in the poly-line. If you make a mistake, you can delete the last point by right-clicking anywhere. Draw the counter sensor so that it is crossed by the vehicles to be counted (see Fig. 5 and Fig. 6).
4. Add a gate (if needed) by clicking the command “Add gate”. A new gate is added in the configuration’s tree view.
5. Draw the gate by left-clicking on the video preview pane. Remember that gates are oriented poly-lines. That is, the order in which you choose the poly-line’s vertexes do matter. The first mouse click defines the gate’s end where vehicles are NOT allowed to enter. Think of it as the tip of an arrow. Draw the gate sensor so that vehicles to be counted move along it towards the arrow tip.
6. Repeat steps 4–5 for each gate you want to add to the probe. Draw the additional gates next to the first gate, with parallel direction.

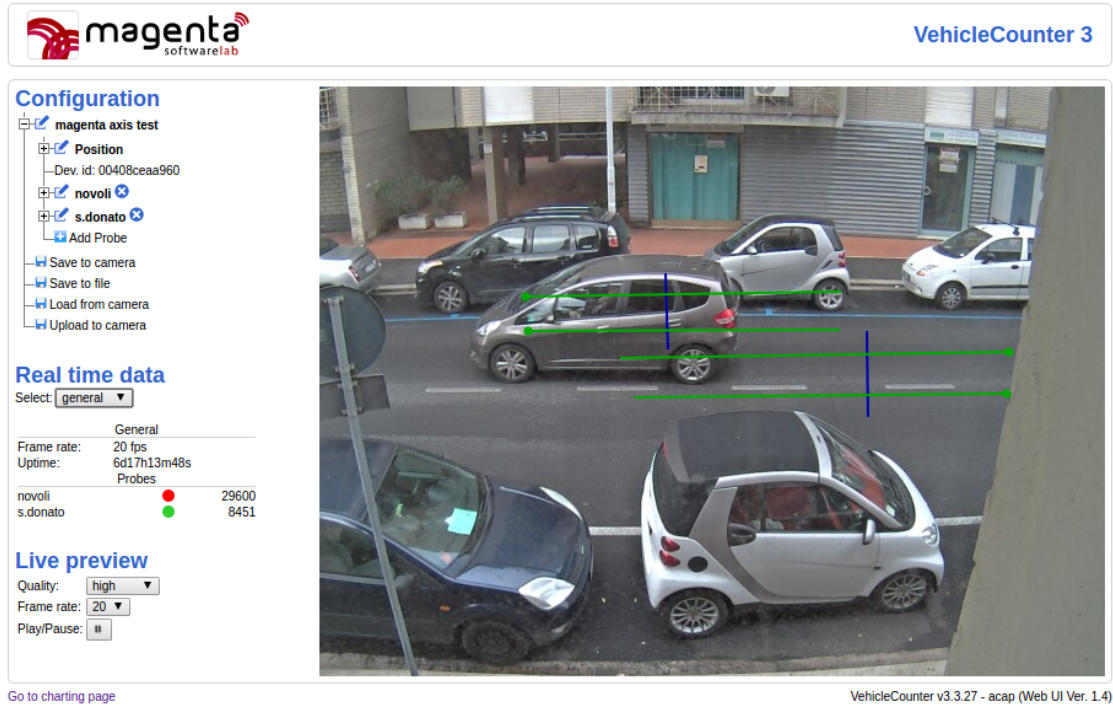


Figure 7: VC3 Configuration page.

7. Save the configuration to camera by clicking "Save to camera"

As soon as the configuration is saved it becomes immediately operative (no need to restart the camera nor the application). Probes and sensors are initialized (initialization of counter sensor may take a while depending on traffic condition) and immediately afterwards the probes start to count.

For a deeper comprehension of how virtual sensors work and how to configure them, see Sec. 13.

7 Diagnostics

VC3 is mainly designed to run unattended for very long time periods. It is supposed to accumulate data in its internal database just for the time that it takes to other pieces of software to get these data from VC3 (through our web API – see Sec. 15). Thus, it is very important to be able to evaluate the performance of VC3 before leaving it to work unattended. To this end, the VC3 configuration page shows some useful diagnostics that helps you detecting problems and issues that need to be fixed.

On the left hand side of the page, right below the configuration's tree view, there is a section called "Real time data" (see Fig. 7 and Fig. 8). A combo-box allows you to select one particular probe, or a summary of all configured probes. By selecting "general", you will be able to see a small text (Fig. 8a) reporting the actual frame rate at which VC3 is working, the uptime, and a table with a summary of information from the configured probes. For each probe the table reports (left to right) the probe's name, a colored icons representing the probe's state (blue:

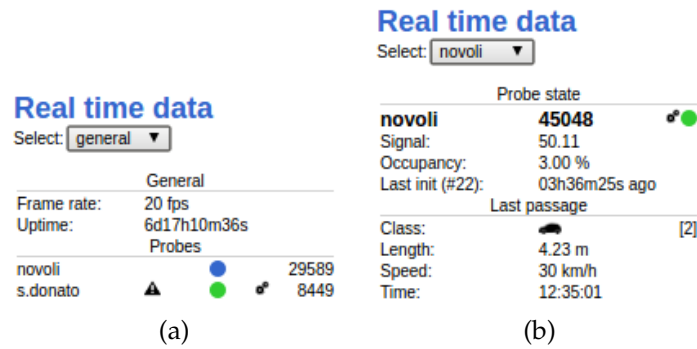


Figure 8: The ‘Real time data’ section. Left: general information. Right: probe’s diagnostic.

idle; green: open; red: closed) and the total number of vehicles detected so far (reset every day). Beside the probe’s name, warning icons for inaccurate length and/or speed estimation and counter sensor’s “busy” state appear when needed.

By selecting a specific probe name in the combo-box, the following informations will show up (Fig. 8b):

- Probe’s real-time data:
 - Probe’s counter and status
 - Value of the signal extracted by the counter sensor
 - Occupancy value
 - Time elapsed since last initialization and number of re-initializations occurred
- Last passage data:
 - Class of last detected vehicle (class icon and class number)
 - Length of last detected vehicle (with warning icon in case the estimate is inaccurate)
 - Speed of last detected vehicle (with warning icon in case the estimate is inaccurate)
 - Time of last passage

All the information shown in the “Real time data” section are updated once a second. Please remember that these data are obtained by sending HTTP requests to VC3. Though the bandwidth needed by this stream of data is extremely low, you may experience delay if your connection with the camera is not stable.

8 Fixing common configuration problems

If VC3 runs with unsatisfactory performance you may want to follow some of these advices to identify the problem and try to fix it.

8.1 Fixing vehicles detection and counting

If VC3 fails in detecting vehicles or counts more vehicle than those actually passed by, there is probably something in your probes configuration that you can change to help things go better.

8.1.1 Vehicle missed

If VC3 fails in detecting some vehicle, check the following.

1. The values of `Upper threshold` and `Lower threshold` parameters may be too high. Check their values and eventually compare them with the values of the signal collected by the probe (see Sec. 7). Default values are 10 and 5 respectively.
2. The counter sensor could be blocked in "busy" state (check if the "gears" icon appears beside the status icon in the "general" information panel). This situation may occur when there is a sudden change in the scene background in an area that intersect with the counter sensor. In this case, wait for the probe to re-initialize the sensor. This should take a few minutes (see `App max busy time in seconds` parameters in Sec. 11). As a hint, remember to periodically check the number of re-initialization in the first few days of operation.
3. If the probe has gates, it is likely that they do not enable the counter sensor because they do not detect motion. It can help to temporarily delete the gates. This turns the probe in a *free probe*, and forces the counter sensor to stay enabled. If it counts vehicles correctly, then there is a problem in the gates. If a gate seems insensitive to motion, check the following.
 - (a) Is the gate correctly oriented? The positive direction of the gate is indicated by the small circle on one end of the poly-line. Objects that enters the gate on the circle's side are "wrong way" vehicles and will not be counted.
 - (b) Gates have a certain degree of insensitivity to slow moving objects. Please note that "slow" refers to the *apparent motion* of objects, not to their actual speed in the real world. A fast vehicle, observed from long distance or with high perspective, may appear slow in the image. In this case, it can help to increase the zoom factor (if possible) or move the camera closer to the monitored area, or to change its point of view so to reduce perspective distortion and get a more clear view of the motion.
 - (c) If a gate is very long, it can intercept static features of the image that are prevalent with respect to that of moving vehicles (e.g. trees, traffic signs, poles). Try with a shorter gate. Also, if `VehicleCounter` is operated behind a window, reflections on the glass may interfere with the actual image and prevent the gates from correctly detect motion.
 - (d) The value of `Frames threshold` and `Delay threshold` parameters affects the gates sensitivity to motion: for both parameters, the higher the value, the lower the sensitivity. The default value for both parameters is set to 1, and that means gates are already *very* sensitive, but if this does not suffice, you can try setting one or both values to 0. We suggest you to try with `Frame threshold` only, at first, and see if this fixes the problem. Note that doing so could lead to a higher rate of false positives.
 - (e) There is one last chance to enable the gates detecting a slow moving object: increase the object speed! Seems unfeasible? It is not! Since from the gates point of view all that matters is *apparent* motion, you can make vehicles *appear* faster by reducing the frame rate in the application settings page! For example, if you reduce the framerate from the default 20 to 10 fps, all moving objects will appear twice as fast. This

will probably solve the problem of gates' insensitivity but may lead to less accurate estimates of vehicles' speed and length, so be cautious.

8.1.2 False positives (double counting)

If VC3 detects more vehicles than that actually observed, then it is probably needed to change the size and/or shape of the counter sensor. If the counter sensor is too small, it can happen that the small portion of the vehicle intercepted by the sensor on one particular frame has too few features to be detected. If this happens, the counter sensor will not be in "busy" state any more, and will consider the vehicle as if it was completely passed. If at a later frame some vehicle's feature is detected, that will be considered as a new vehicle and will be counted as such.

This behaviour is not so common, but it can happen if lighting conditions are poor and counting sensors are small. Lighting conditions often cannot be controlled, so the only thing we can do is to increase the size of the counter sensor and/or its shape. A few examples of complex shapes are shown in Fig. 5.

8.2 Fixing length and speed estimation

Length and speed estimation requires accurate vehicle detection and counting, properly configured gate sensors, correct value of the relevant parameters, and favourable conditions of operation. First of all, length and speed estimation is only achievable via gate sensors. Gates in a probe in fact are not only used to enable the counting sensor, but to estimate the vehicle's speed in the gate's positive direction. By knowing the approximate speed of a vehicle in every frame that it takes to cross the counting sensor, the probe can also estimate the vehicle's length. However, vehicle's speed and length are originally computed in $\frac{pixels}{frame}$ and *pixels* respectively. These measures must be translated in the usual metric units to become meaningful. To do this, it is required that the user provides some reference measure, e.g. the gate's "apparent length" in centimeters. This allows the application to turn the length estimate in centimeters and the speed estimate in Km/h.

To achieve a reasonably correct value of the gate's length, the best way is to find an example with one (or more) car model of known length. The preview panel in the VC3 configuration web page can be paused on a particular frame taken from the live stream. By pausing the live stream when a vehicle of known length is moving along the gates, it should be easy to assign the gates a reference length by comparison.

It must be pointed out that the speed and length measures provided by VC3 are approximated, and that their accuracy is strictly related to the condition of operations, degrading more and more as the perspective in the scene increases. However, accuracy can be increased by using more than one gate in a given probe. In this way, vehicle's speed and length will be computed by averaging the measures provided by all the gates involved. Also, using more than one gate makes it unlikely to miss a vehicle because it has not intercepted the gate sensor.

Another aspect that have an effect on the accuracy of measured speed and length is the zoom factor. In fact, it controls the ratio of centimeters against pixels and obviously the amplitude of apparent motion of moving objects. If the apparent motion of vehicles is too small, increasing the zoom factor (or, moving the camera closer to the scene) may help VC3 to work better.

Last but not least, the gate size may affect VC3 performance. If the gates are too short, they may fail in filtering wrong-way vehicles, and that would bring inaccurate counting. If the gates are too long, they could detect movements occurring far from the counting sensor, enabling it too early and thus collecting possible false positives. Also, very long gates are more likely to intercept static features from the background that could be prevalent with respect to non-static features from vehicles, making them less sensitive to motion. In general, a rule-of-thumb for sizing the gates is to make their length comparable with that of the vehicles to be counted.

9 How to get the data

Each time VC3 detects a vehicle, it writes a line in a table of an internal SQLite3 database, recording all relevant data for that passage. Thus, you can have the finest level of granularity: each passage is recorded as a single timestamped event, to which several information are attached: some describe the probe that detected the vehicle, while others describe the vehicle itself (see the description of `getRawData.cgi` in Sec. 15). Collected data are not stored indefinitely. To prevent storage space shortage, they are deleted after a configurable amount of days (`App memory size in days` parameter), which defaults to 7. If you want to build your own historical database to which send traffic data, or simply want to conduct analysis on the collected data, you have to query the data from VC3 before they are deleted from the internal DB.

There are two methods to extract data from VC3: *on-demand* data transfer, or automatic data transfer.

9.1 On-demand data transfer

On-demand data transfer requires you to perform HTTP requests to VC3, invoking a specific CGI and passing a few parameters to select the data of interest. Your web browser will work, for example, but you may use whatever tool you find suitable for this purpose.

VC3 provides a quite complete set of CGIs. You can find a detailed list in Section 15. However, traffic data extraction are usually achieved invoking `getRawData.cgi`, like in this example (copy it in your browser address bar and replace `<camera-addr>` your camera IP address):

```
http://<camera-addr>/local/VehicleCounter/getRawData.cgi?interval=3600
```

VC3 will respond with a JSON document containing a vector of all passages detected within the last 3600 seconds (1 hour). Please note that since the `interval` value is applied with respect to the time VC3 receives the request, repeating the same request after a few seconds usually does NOT give the same result (because older passages may have moved outside of the time interval that has been queried). However, most of the data will be repeated in the new response, if the elapsed time is short enough. If this is not desirable, it can be avoided by adding to the request the `once` parameter:

```
http://<camera-addr>/local/VehicleCounter/getRawData.cgi?interval=3600&once=
```

This request does exactly the same thing as the previous one, plus:

- it marks all extracted passages with a special flag, to state that they have already been queried at least once before;
- it discards all passages in the selected time interval that have the special flag already set.

So, if the second sample request is repeated, let's say, every minute, you will only get one minute of data each time (the latest), because older passages have already been extracted in previous requests. Only the first query would actually return an entire hour of data, but subsequent queries would discard most of data in the selected time interval because they have the special flag set from previous queries.

This way of performing on-demand data transfer allows you to repeatedly acquire data with the desired frequency, while maintaining a certain tolerance to network and transfer failures: if at a given time, a request cannot complete for any reason, newer data will not have their flag set, but they will be included in the response to the query the next time it is requested. This is the recommended way to perform on-demand data transfer for feeding an external data storage, so to avoid data duplication. Ensure to choose wisely both the value of `interval` parameter and the frequency of the request, as tolerance to failures strictly depends on their values.

9.2 Automatic data transfer

Automatic data transfer has been introduced in version 3.3 to ease integration with some external data storage platforms. It allows you to instruct VC3 to periodically send data to a specific address, assuming that there is a CouchDB server listening for PUT requests at that address. The relevant parameters you need to set are the following (see Sec. 11):

`Couchdb period in minutes`: Frequency of automatic data extraction and transmission towards an external Couchdb server. If set to 0 disables automatic data extraction.

`Couchdb url db name`: name of the database that will store the transmitted data. This database must exist in the external Couchdb server to which the data will be transmitted.

`Couchdb url host`: Domain name or IP address to which data will be transmitted.

`Couchdb url port`: Port number of the external Couchdb server.

`Couchdb auth password`: User password (for non-public database).

`Couchdb auth username`: User name (for non-public database).

`Couchdb https`: If checked, uses HTTPS instead of HTTP during data transfer.

When `Couchdb period in minutes` is set to a value greater than 0, VC3 will start performing data extraction using a query equivalent to the following:

```
http://<camera-addr>/local/VehicleCounter/getRawData.cgi?interval=<time>&once=
```

where `<time>` is the equivalent in seconds of five times the requested transmission period. So, if `Couchdb period in minutes` is set to 2, the value `<time>` will be $2 \times 5 \times 60 = 600$ (seconds). This should provide tolerance to network failures that last no longer than 10 minutes.

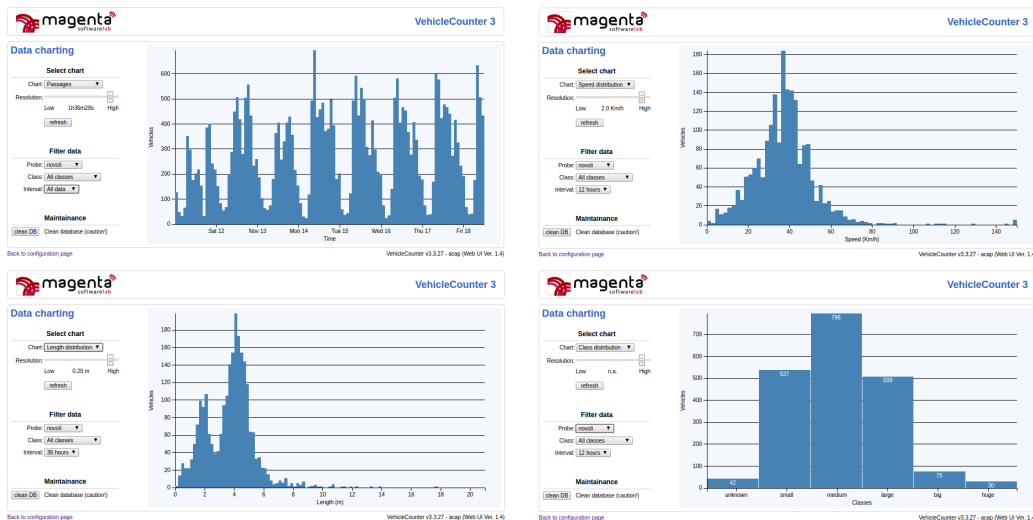


Figure 9: Data charts. From top–left corner, clockwise: passages chart, speed distribution chart, length distribution chart, class distribution chart.

10 Data charts

The link in the lower left corner of the configuration page (Fig. 7) leads to VC3 “Charting page”. There you can visualize collected data in graphical form, and this is of great help both for validating the configuration in early post-installation phase, and for a first analysis of the traffic flows.

The page allows you to plot four different types of charts (Fig. 9):

- **Passages:** an histogram of the vehicles detected in the selected time interval
- **Speed distribution:** distribution of detected vehicle’s speed in the selected time interval
- **Length distribution:** distribution of detected vehicle’s length in the selected time interval
- **Class distribution:** distribution of detected vehicle’s class in the selected time interval

On all charts, the page let the user choose:

- for which probe to extract the data
- for which class to extract the data
- the ‘interval’ value: this has the same meaning than the `interval` parameter in the `getRawData.cgi` request

Data charting can be of great help in validating the sensors configuration. Distribution charts, particularly, may suggest the existence of problems such us wrong speed and/or length estimates or misclassifications, enabling you to take countermeasures.

Note that the charting page also allows to clean the internal DB, so that it is possible to start a new campaign of data acquisition with an empty database.

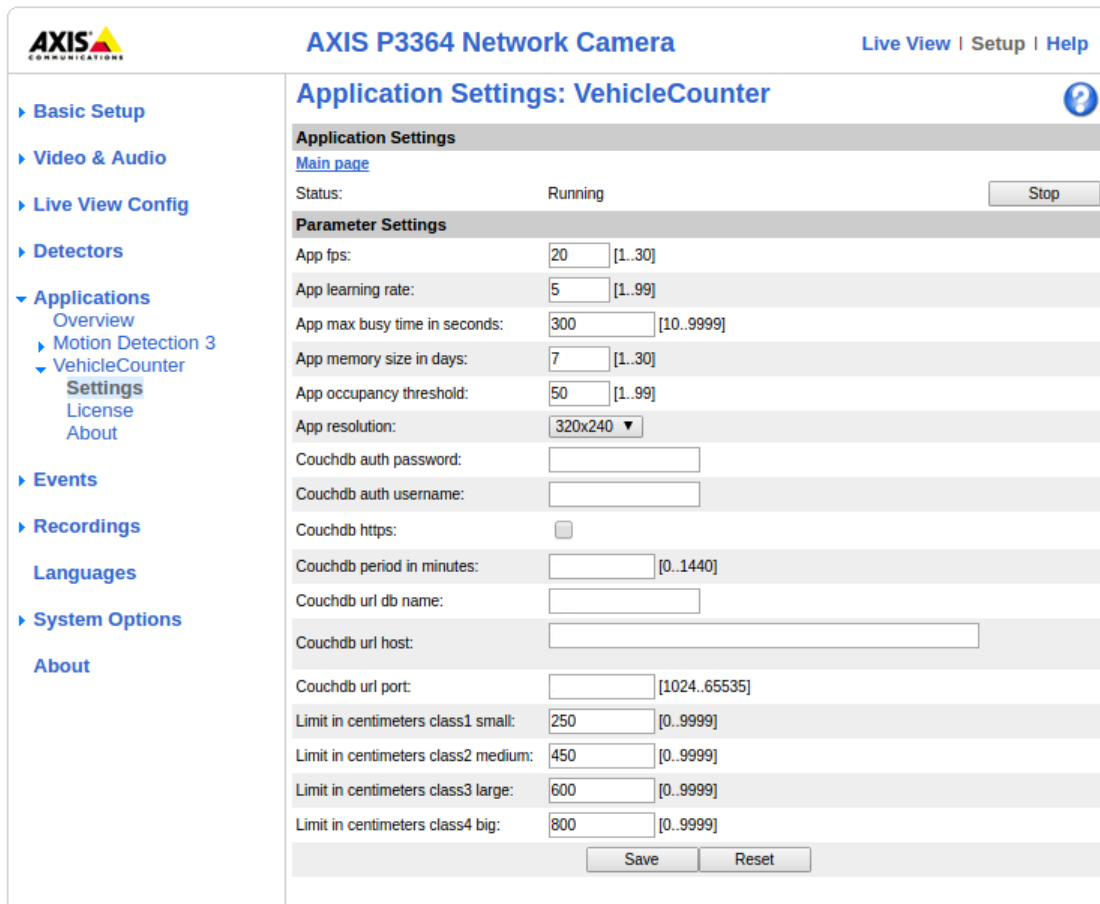


Figure 10: The 'Application Settings' page

Part III

Comprehensive guide

11 Application parameters

The "Settings" page (see Fig. 10) allows the user to provide a set of parameters that controls a few features of VC3. Remember to click the "Save" button and to stop and restart VC3 whenever you change some parameter's value. The configurable parameters are the following.

App fps: the frame rate at which the application grabs frames from the sensor. Keep in mind that frame grabbing on the camera is an independent process. Grabbing frames at a frame rate higher than that supported by the application will reduce the performance. We recommend not to change this unless you really understand what you are doing. Default value: 20. Maximum value: 30.

App learning rate: counter sensors learn a background model during probe's normal operation. The parameter's value is the (percentage) learning rate of the background learning process. High values of this parameter will make the learning process very fast, while low

values will slow it. We recommend no to change this unless you really know what you are doing. Default value: 5.

App max busy time in seconds: maximum number of seconds that a counter sensor is allowed to spend in 'busy' state before a re-initialization is triggered. Default value: 300.

App memory size in days: defines how many days of data VC3 must retain. Data clean up is performed every day at 23:59 and deletes all data older than this amount of days. Default value: 7.

App occupancy threshold: Maximum value allowed for the sensor occupancy (percentage). When the actual occupancy raises above this threshold, an alarm is triggered. Default value: 50.

App resolution: resolution of the image grabbed by the application. Higher resolution means the application will process more pixels and this has a strong effect on the performance. The typical reason why you may want to change this is not to increase the resolution, but to change the image aspect ratio, since this may provide you a wider field of view (depending on the camera model). The default is 320x240, but you may pick one from this list (recommended values are in boldface):

4:3 aspect ratio: 640x480, **320x240**, 160x120

16:9 aspect ratio: 640x360, **320x180**, 160x90

CIF aspect ratio: 704x576, **352x288**, 176x144

Please note that not all these resolutions and/or aspect ratios are available on all Axis camera models. Before changing `App resolution` please ensure your camera can handle the new resolution value.

Couchdb auth password: If you set up automatic data transfer to a non-public Couchdb database, write here your password.

Couchdb auth username: If you set up automatic data transfer to a non-public Couchdb database, write here your user name.

Couchdb https: If checked, enables the use of https during the automatic data transfer to a Couchdb database.

Couchdb period in minutes: Frequency of automated data extraction and transmission towards an external Couchdb server. Default value is 0 (no transmission).

Couchdb url db name: name of the database that will store the trasmitted data. This database must exist in the external Couchdb server to which data will be transmitted.

Couchdb url host: Domain name or IP address of an external Couchdb server to which data will be transmitted.

Couchdb url port: Port number on which the external Couchdb server is listening for PUT requests.

Limit in centimeters class 1 small: maximum length in centimeters allowed for class 1 (motorcycles and small cars). If the estimated length exceeds this threshold, the vehicle will be classified in one of the remaining four classes of vehicles.

Limit in centimeters class 2 medium: maximum length in centimeters allowed for class 2 (medium cars, economy cars). If the estimated length exceeds this threshold, the vehicle will be classified in one of the remaining three classes of vehicles.

Limit in centimeters class 3 large: maximum length in centimeters allowed for class 3 (large cars or vans, sedans). If the estimated length exceeds this threshold, the vehicle will be classified in one of the remaining two classes of vehicles.

Limit in centimeters class 4 big: maximum length in centimeters allowed for class 4 (small trucks). If the estimated length exceeds this threshold, the vehicle will be classified in class 5, reserved to large trucks, buses and other large vehicles.

12 VehicleCounter configuration page

The VehicleCounter configuration page is accessible by clicking the "Main page" link on the "Settings" page (see Fig. 10).

As shown in Fig. 7, the configuration page is split in two. On the right hand side, a preview panel shows the camera video stream with the desired quality and frame rate. Note that the preview panel has a fixed width of 640 pixels regardless of the working resolution (App resolution parameter) and of the preview panel quality. The height will be fixed according to the aspect ratio of the current working resolution (4:3, 16:9 or CIF).

Currently configured virtual sensors are drawn on the preview panel as poly-lines superimposed on the camera image. Counter sensors are drawn in blue, while *standard* gate sensors are green. Since version 3.3, there exist two more types of gate sensor: *mandatory* gates, drawn in orange, and *safeguard* gates, drawn in red. See Section 13.5 for an explanation of the differences. The 'current sensor' (either counter or gate) is highlighted both in the preview panel and in the tree view.

On the left hand side of the page are placed several controls divided in three sections. From top to bottom:

- **configuration tree view:** shows the current configurations of probes, with parameters for the configuration, the counting sensors and the gates. Contains controls to add or delete probes and gates, to change the parameters' value, to save the configuration to camera or to file and to reload the configuration from the camera. Configuration file upload is also possible.
- **real time data:** shows relevant diagnostic information about the application and the configured probes (updated once a second when active).
- **live preview:** contains controls to change the live preview image quality and frame rate, as well as a play/pause button to stop the live stream on a particular frame (fast reflexes needed!)

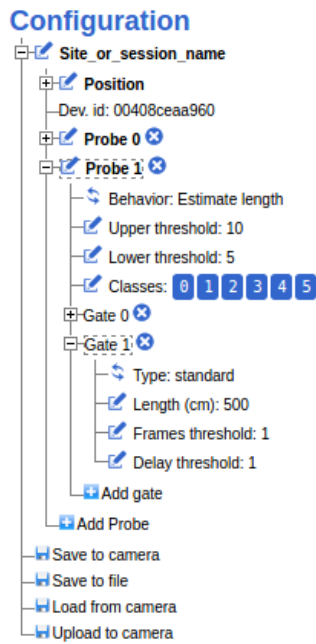


Figure 11: The configuration's tree view.

Icon	Meaning
	Changes the name of the probe/gate or the parameter's value
	Cycles through the possible values of a parameters
	Deletes an entire probe or removes a gate from a probe.
	Adds a probe or a gate to the current configuration.
	Transfer the current configuration to/from the camera or a file.

Table 1: General meaning of the icons used in the configuration's tree view.

12.1 Configuration tree view

In the configuration tree view it is shown the current probes' configuration, as well as controls to add and remove probes and gates, change the value of various parameters, save the configuration to camera or to a file, re-load the configuration from the camera and upload a configuration file to the camera (see Fig. 11).

Hovering tips will help you get confident with the use of our web interface, however Tab. 1 shows the general meaning of the icons used in the tree view, and Tab. 2 shows a comprehensive list of the available controls, with a brief explanation. Deeper details are provided, when needed, in other sections of this user manual.

Please note that the wording "current configuration" here refers to the configuration shown on your web browser, NOT the configuration actually in use in the VC3. When you load the configuration page in your browser, it will ask VC3 for a copy of the configuration it is using, and show it on the tree view and on the preview pane. Then you can start making changes to the configuration, but this changes remain local to your browser until you click "Save to camera". This will transfer the current configuration to your Axis camera and to the VehicleCounter application, which will parse and apply it, if valid.



















 Site_or_session_name	Assigns a name to the current configuration. Choose a meaningful name, for example the name of the street or the address of a nearby house.
 Position	Sets GPS coordinates of the camera on a map. Click the '+' icon to show separate controls for latitude and longitude
 Probe 0	Changes the name of probe with id 0. Use meaningful names, for example the direction of vehicles in terms of North, South, East or West.
 Upper threshold: 10	Changes the value of "Upper threshold" parameter. It must be greater than the value of "Lower threshold" parameters and defines the level of the signal above which the counter sensor becomes "busy".
 Lower threshold: 5	Change the value of "Lower threshold" parameter. It must be lower than the value of "Upper threshold" parameters and defines the level of the signal below which the counter sensor becomes "idle".
 Length (cm): 500	Sets the (apparent) length of a gate in centimeters.
 Frames threshold: 1	Sets the value of "Frames threshold" parameter in a gate. It has effect on gate's sensitivity to motion. The higher the value, the lower the sensitivity. Minimum value: 0.
 Delay threshold: 1	Sets the value of "Delay threshold" parameter in a gate. It has effect on gate's sensitivity to motion. The higher the value, the lower the sensitivity. Minimum value: 0.
 Behavior: Estimate length	Changes the behavior of a probe regarding length estimation of vehicles. Possible values are "Estimate length" (default) and "Fixed length".
 Type: standard	Changes a gate's type. Possible values are "standard" (default), "mandatory" and "safeguard".
 Classes: 	Changes a probe's class mask. Clicking a class number will enable/disable it. Clicking on the icon will enable/disable all classes. Vehicles that fall in a disabled class are not counted, nor they trigger any event.
 Add Probe	Adds a new probe to the configuration tree view and selects it. Left/right clicks on the preview panel will add/remove points to the counter sensor.
 Add gate	Adds a new gate to the current probe and selects it. Left/right clicks on the preview panel will add/remove points to the gate sensor.
 Save to camera	Saves the current configuration to camera. Once received, the camera will load the new configuration and re-initialize the probes. If no error occurs, the new configuration will immediately become operational, otherwise the previous configuration is restored.
 Save to file	Saves the current configuration to a local JSON text file.
 Load from camera	Re-loads probe's configuration from the camera. Be careful, as this will cancel all unsaved configuration changes.
 Upload to camera	Uploads a previously saved configuration file to the camera (the file must be a valid VehicleCounter configuration in JSON format).

Table 2: A complete list of the controls available in the configuration's tree view.

12.2 Real time data

This section shows many useful diagnostic information about the system and the probes currently in use. A combo box allows the user to choose between the following choices:

- "-": Does not display any information and stops all current CGI requests.
- "general": Shows general informations about the application and a summary of the most important informations from the probes (see Fig. 8a).
 - **Frame rate**: the actual frame rate at which VC3 is running. In some cases, this number can be lower than the requested frame rate (see application parameters). If the difference is more than 1-2 frames, than VC3 is probably spending more time trying to acquire frames than to process them. Consider the possibility to reduce the frame rate in the application parameters page. That could make the actual frame rate increase a bit.
 - **Uptime**: the total time elapsed since the application started.
 - Summary of probe's information: for each configured probe a line of text is added showing (left-to-right): the probe's name, eventually a warning icon, the probe state (see Tab. 3) and total number of counted vehicles.
- <ProbeName>: Shows specific information about the probe's state and the last detected vehicles. Particularly (top-to-bottom and left-to-right, see Fig. 8b):
 - the probe's name, total number of counted vehicles and probe's status (see Tab. 3).
 - the current 'level of signal': this signal is what VC3 uses to detect vehicles. The `Upper threshold` and `Lower threshold` parameter apply on this value, that is, if this value raises above the `Upper threshold` value, the sensor becomes 'busy'; if this value falls above the `Lower threshold` value, the sensor becomes 'idle'.
 - the occupancy percentage. Every 60 seconds `VehicleCounter` computes the time that was spent by the probe in 'busy' state in the last 5 minutes. If this ratio (in percentage) is higher than the value of `max busy time in seconds` parameter, an alarm is triggered.
 - number of (automatic) re-initialization and time elapsed since last re-init. A counter sensor is not allowed to stay in 'busy' state indefinitely. After a given amount of time (see application parameters) the probe is re-initialized to prevent the application from stopping when some unusual event occurs in the scene. These information are useful to understand how frequently this kind of events are occurring and eventually decide to take countermeasures.
 - class of the last detected vehicles. Vehicles are assigned one out of five classes based on their estimated length. Classes are represented by an icon (see Tab. 4). The boundaries between the five classes may be set in the application parameters' page.
 - length (in meters) of last detected vehicle. A warning icon is also shown if VC3 detects particular events that could make the provided measure inaccurate.






icon	description
	idle (gated probes only)
	open
	closed (gated probes only)
	busy (vehicle on sensor)
	initializing

Table 3: The available probe’s status and indicators.






icon	description	class id
-	Unknown class	0
	Bikes, motorcycles and small cars	1
	Medium cars (economy cars)	2
	Large cars (sedans) and vans	3
	Trucks, light-duty vehicles	4
	Large trucks, buses	5

Table 4: The available vehicle classes.

- speed (in Km/h) of last detected vehicle. A warning icon is also shown if VC3 detects particular events that could make the provided measure inaccurate.
- time of last detected passage.

12.3 Live preview

In this section are grouped the controls to operate on the live preview panel. Proceeding top-to-bottom:

- **Quality:** this combo-box allows to choose amongst three different level of image quality: ‘high’, ‘medium’ and ‘low’. Lower image quality means less bandwidth needed to transmit the image for the preview pane. The default value is ‘medium’.
- **Frame rate:** the frame rate defaults to 1 so to save bandwidth in case you are accessing the camera via a wireless link, but you can raise it up to 20 fps.
- **Play/Pause:** this button pauses the live stream allowing you to work on a still frame. This is particularly useful when you have to tune the ‘length’ parameter for a gate sensor. Since the exact length is not measurable on the ground, the best thing to do is to use an example, i.e. find a car of known length and see how it appears to the camera, taking into account lighting conditions, perspective etcetera. You can do this by pausing the live video stream at the right time, so to capture a frame that shows a particular car model of known length. Since the probes are drawn superimposed on the live video, it should be easy to assign a length to the gate by comparing it to the vehicle.

Please note that the resolution and frame rate you choose in the ‘Live preview’ section do not affect VC3 performance. The actual resolution and frame rate of images processed by VC3 are controlled by the parameters `App resolution` and `App frame rate` in the application settings page.

13 Configuration guidelines

13.1 General principles

The VC3 configuration is basically a list of *probes*. A probe is made of a counter sensor, and optionally one or more gate sensors. All sensors are poly-lines that can be drawn directly upon the image grabbed by the camera, using the "VehicleCounter Configuration" page. Gate sensors also have a *positive direction*, which defines the direction that vehicles must follow in order to be allowed into the counter sensor. Visually, the positive direction is the one that goes along the sensor, toward the little circle placed at one end. If the probe has at least one gate, it is called *gated probe*, otherwise it is a *free probe*. A free probe is always enabled, meaning that its counter sensor will count whatever moving object crosses it, in any direction, except when the probe is in its initialization phase (that usually takes a few seconds) or when the probe is "busy" because the last detected vehicle has stopped on the sensor. A gated probe, instead, has in general its counter sensor disabled. In order for the probe to count a vehicle, it is needed that the vehicle goes through the gates in their positive direction with a perceivable velocity. If this happens, the gates detect the motion and enable the counting sensor, allowing the vehicle to be counted. Note that the gate will disable the counting sensor when it stops detecting motion, so the counting sensor must be placed approximately half-way along the gate sensor. A gated probe is likely to have more than one gate. In this case, they are usually placed very close to each other with parallel positive directions. In this kind of probe, the counting sensor is enabled if at least one of the gates detects motion in its positive direction.

Since version 3.3, VehicleCounter has added other features both to probes and gates. Details on how to use these new features will be given in Section 13.3 and Section 13.5. However, this is a brief summary:

- Each probe can now operate in one out of two possible *behaviors* that differs by the way they deal with length of detected objects: *estimate lenght* estimates vehicle length as usual (this is the standard behavior and the default parameter value), while *fixed length* only estimate objects' speed, assuming they have fixed length. This can be useful when counting small quasi-uniform objects that are hardly separable, like pedestrians in a crowd.
- It is now possible to disable some vehicle classes, so to make a particular probe ignoring all vehicle of certain classes.
- A gate can now be of one out of three different types: *standard*, *mandatory* or *safeguard*, that differs by the way they affect the counter sensor they are attached to.

13.2 Drawing the counter sensor

As shown in Fig. 5, a counter sensor can have practically any shape, ranging from the simplest rectilinear segment to the most complex poly-line. The difference between the different shapes are essentially the following:

- Sensors with complex shapes are more sensitive. If you observe a few false negatives (i.e. missing counts), using a more complex shape as a counter sensor may solve the problem.

- Longer counter sensors are less sensitive. Keep this in mind when you draw complex-shaped counter sensors, as they can easily get longer than simple ones.
- Large sensors may miss the gap between two consecutive vehicles. VC3 detects a vehicle in all the frames that it takes to cross the counter sensor, from when it enters to when it leaves. If the counter sensor spans amongst a wide area of the image frame, it may happen that two consecutive vehicles are seen like one because when the second vehicles enters the sensor, the first one has not left the sensor yet. In this case, you would experience a missing count and a bad length estimate. To avoid this problem prefer thin shapes and keep perspective into account.

In *gated probes*, counter sensors can be drawn with exactly the same guidelines in mind, but the presence of gate sensors requires to take into account a few other things:

- The counter sensor has to be positioned in such a way that it is impossible for a vehicle of interest (i.e. a vehicle that we want the counter sensor to detect):
 - to cross the counter sensor without having intercepted at least one gate before;
 - to leave the gates without having intercepted the counter sensor.
- The best position for the counter sensor with respect to the gates is approximately halfway the gate length, as in Fig. 6. If the counter sensor is too close to the gate's entrance it may happen that vehicles enter the counter sensor before the gates have detected the vehicles' motion, and this would lead to inaccuracy in the estimation of the vehicle's length (and consequent vehicle's misclassification), or even to missed vehicle's detection.

13.3 Counter sensor's parameters

Counter sensor's parameters are actually probe's parameters: since a probe has always exactly one counter sensor, we usually refer these parameters to the probe instead of the counter sensor, and particularly in the configuration tree view, where you can act on the following:

Upper threshold: this parameter, together with **Lower threshold**, controls the sensitivity of the counter sensor to moving objects. During operations, the counter sensor continuously extract a signal from the image. You can see it in the "Real time data" section by displaying a specific probe status ("Signal level"). The counter sensor goes in 'busy' state (i.e. it detects a vehicle) when the signal level raises above **Upper threshold**. Its value must be (strictly) greater than that of **Lower threshold**, as this will help coping with natural fluctuations of the signal. The higher the value, the lower the sensitivity. The default value is 10.

Lower threshold: this parameter, together with **Upper threshold**, controls the sensitivity of the counter sensor to moving objects. It defines the value of the signal below which the counter sensor goes back into 'idle' state from the 'busy' state (i.e. the vehicle has passed the counter). Its value must be (strictly) lower than that of **Upper threshold**, as this will help coping with natural fluctuations of the signal. The higher the value, the lower the sensitivity. The default value is 5.

Behavior: changes the behavior of a probe with respect to the length estimation of detected moving objects. This has only effects on *gated probes* that have at least one standard or mandatory gates. There are two possible values:

estimate length: (default) this is the standard behavior for a gated probe. When a vehicle is detected, triggered gates estimate its speed and length, and based on these information the vehicle is assigned to a class.

fixed length: uses gates to only estimate speed, and assumes that moving objects have fixed length (equal to the length set in gates' `length` parameter). Based on the speed and the time spent by the counter sensor in 'busy' state, the probe will try to guess if the counter was triggered by a single object or by a 'blob' of objects, and in this case will estimate the cardinality of the blob (that is, how many objects compose the blob). In other words, a single trigger of the counter sensor may lead to multiple counts. This can be useful when counting small quasi-uniform objects that are hardly separable, like pedestrians in a crowd. Objects counted this way are always assigned to class 0 ("unknown").

Classes: the list of vehicle classes that must be counted by the probe. This allows the user to disable (*mask*) some classes. Vehicles that falls into a disabled class will be ignored: their passages will not be written in the database, and they will not trigger any event.

When modifying `Upper threshold` and `Lower threshold` parameters, keep into account that when the sensor is idle the typical signal value is bounded in $[0, 1]$. The maximum values reached depends instead from the particular camera setup. Check the value displayed in the "Real time data" to get an idea of the range of values spanned by the signal.

13.4 Drawing the gates

A gate sensor is a poly-line with a direction associated. That direction specifies how a vehicle is supposed to go through the poly-line in order to be detected by the sensor, thus one could say that one of the gate's end is the *entrance* and the other is the *exit*. The gate's exit is visually marked by a small circle. Vehicles moving along the gate from the entrance toward the exit allow it to enable the counter sensor, so that when it is crossed by the vehicle, it can detect it.

Gates should be drawn with the following guidelines in mind.

- Gates are always drawn from the exit to the entrance.
- Gate's length should be comparable with the length of vehicles on the image frame.
- In a multi-gate probe, gates are typically drawn close to each other, with parallel positive directions (this may vary in some situation, though)
- It is preferable to draw gates in poorly textured regions, whenever it is possible

It is worth to note that gates are sensitive to *apparent motion*. Based on the camera point of view, perspective can make fast moving vehicles look very slow. This can prevent the gate from detecting its motion and thus the counter sensor from detecting (and counting) the vehicle. To

fix this kind of problems it can be useful to vary the camera focal length, or physically move the camera closer to the point where vehicles are to be counted (if possible). This way vehicle's motion will appear amplified in the image and thus easier for the gates to detect.

Unfortunately, often camera position and focal length cannot be freely modified. However, there are something that can be done to make vehicles *appear* faster in the image:

- By reducing the frequency at which the camera samples the scene, all objects will appear faster. Thus, reducing the application frame rate (App `frame rate` parameter) from the default 20 fps to, say, 10 fps, all moving objects will appear twice as fast, and this will make them more detectable by the gates. As a drawback, this could reduce accuracy in speed and length estimation, and consequently may lead to a higher rate of misclassified vehicles.
- Increasing the application working resolution (App `resolution` parameter) will increase the ratio of pixels against centimeters in gates length, thus a moving object will appear faster in terms of its $\frac{\text{pixels}}{\text{frame}}$ velocity. As a drawback, this will also increase the number of pixels to be processed and consequently the CPU load, eventually lowering the frame rate.

13.5 Gate sensor's parameters

Each gate sensor has the following parameters that can be tuned using the controls embedded in the configuration's tree view.

Type: Since VC3.3 there exist three different types of gate. These are the available types:

standard: the simplest, usual gate. Everywhere in this manual, if not stated otherwise we assume a gate has this type. Standard gates forces the counter sensor to be disabled until at least one of them detects motion in its positive direction. When this happens, all standard gates that detects motion contribute to the vehicle's length and speed estimate. Note that if a standard gate detects motion in the wrong direction, this does not prevent the counter sensor from counting a vehicle, provided that at least one other standard gate detects motion in its positive direction. Standard gates are drawn in green on the preview pane.

mandatory: Mandatory gates behaves exactly as standard gates, except that they *must* be triggered by the moving vehicles in order for the counter sensor to be enabled. In other words, the counter sensor will not be enabled unless all its mandatory gates detect motion in their positive direction. This can be useful to ensure that all vehicles counted by a particular probe are moving on a specific image region. Mandatory gates are drawn in orange on the preview pane.

safeguard: Safeguard gates are very special gates, as they behave differently from standard and mandatory gates. In fact, safeguard gates do not normally disable their counter sensor. This means that a probe with only safeguard gates behaves just like a free probe, because vehicles are detected even if they do not trigger the gate in its positive direction. But, if a safeguard gate detects motion in its wrong direction, then

the counter sensor is disabled and this prevents it from counting until none of the safeguard gates detects motion in its wrong direction any more. This gate can be useful in complex situation where it is needed to reduce the rate of false positives, either on free or on gated probes. Safeguard gates are drawn in red on the preview pane.

Length: this is probably the most important amongst gate's parameters. it defines the ratio $\frac{cm}{pixels}$ that allows to turn velocity estimation from $\frac{pixels}{frame}$ to $\frac{m}{s}$. For hints on how to tune this parameter check Sec. 5.

Frames threshold: in order to reduce false positives, gates will consider detected motion only if it is observed for a number of frames greater than this value.

Delay threshold: in order to reduce false positives, gates will consider detected motion only if its magnitude is greater than this number of $\frac{pixels}{frames}$.

14 Triggering events

When hosted by an Axis camera, VC3 can be used to transmit notifications to other devices if particular events of interest occur. These events are:

- vehicle detected in the positive direction of a given probe;
- vehicle detected in the wrong direction of a given probe;
- still vehicle detected on a probe's counter sensor;
- heavy traffic detected on a given probe.

VC3 integrates perfectly with the Axis camera events system. When one of the above situations occur, it triggers an event that can be catch by the Axis camera's on-board events system, allowing the user to perform proper actions in response. For example, send an e-mail or an SMS, start recording a video, send images to a network share or drive an external device (only if your Axis camera has I/O ports).

While the application is running, the following events appear as triggering events for the VehicleCounter application:

- `count-<probeId>`: triggered when the probe `<probeId>` counts a vehicle; note that a vehicle is counted (and the corresponding event is triggered) when the counter sensor goes back to 'idle' from the 'busy' state (i.e., when the vehicle leaves the sensor);
- `wrongway-<probeId>`: triggered when the probe `<probeId>` detects a vehicle in the wrong direction;
- `stillvehicle-<probeId>`: triggered when the probe `<probeId>` detects a still vehicle on its counter sensor;
- `heavytraffic-<probeId>`: triggered when the probe `<probeId>` occupancy value raises above `Occupancy threshold` parameter (occupancy value is checked once a minute).

To be able to catch one of these events you will need to set up a new Action Rule. Go to the Events menu, Action Rules and click the "Add..." button. In the "Condition" section, select Applications in the first combo-box. A second combo-box will default to "VehicleCounter" and a third combo-box will appear, allowing you to select the event of interest.

The available actions are listed in the "Type" combo-box in the "Actions" section and they usually depends on the particular camera model you are using. Please refer to your Axis camera's documentation for details.

15 Web API

VC3 stores data in an internal SQLite3 database. For each detected vehicles, a record is written in a database table. The collected data can be queried via a web service API that allows you to get them with different levels of filtering and aggregation. Requests must be sent to the address:

```
http://<camera-ip>/local/VehicleCounter/<API-name-and-parameters>
```

Most of the APIs respond with a JSON document storing the requested informations, and usually the responses contain the following common fields:

"build-version": VC3 build version

"cameraId": dumps the MAC address of the VC3 host

"query": echoes the name of the CGI that handled the request

"localtime": local date and time at which the request was handled

"gps-lat": latitude of the camera (as set in the VC3 configuration)

"gps-lon": longitude of the camera (as set in the VC3 configuration) by the application

Most APIs accept optional parameters. In this case, the parameters value is echoed in the response, thus the response MAY contain the following fields:

"interval": if the invoked CGI accepts the interval parameter, its value is dumped here.

Note that the dumped value is the actual time interval used in the query, which may differ from the value used in the request. For example, if the request was made with a `interval=3600` but the application was only started since 30 minutes, the response will show `"interval": 1800`. If the parameter is optional for the invoked CGI and was omitted in the request, the response will show `"interval": null`.

"probe": if the invoked CGI accepts the pid parameter, its value is dumped here. If the parameter is optional for the invoked CGI and was omitted in the request, the response will show `"probe": null`.

"class": if the invoked CGI accepts the cls parameter, its value is dumped here. If the parameter is optional for the invoked CGI and was omitted in the request, the response will show `"class": null`.

"resolution": if the invoked CGI accepts the res parameter, its value is dumped here. If the parameter is optional for the invoked CGI and was omitted in the request, the response will show the default value, which depends on the particular API called.

Beside those intended to query data from VC3, there are several APIs designed to carry out specific tasks, such as:

get probe's configuration: `getConfiguration.cgi` returns a JSON with the entire VC3 probe's configuration;

set probe's configuration: `setConfiguration.cgi` can be called with POST method to send a new configuration to VC3. The body of the request must contain the new configuration in JSON format.

get the VC3's local time: `getLocalTime.cgi` will respond with the UTC time of the VC3's host, expressed in seconds since epoch.

set the VC3's local time: `setLocalTime.cgi` can be used to request VC3 to set a new value for the internal clock. The new time must be transmitted as value of the time option, either in GET or in POST mode. The new time is intended UTC, expressed in seconds since epoch.

get a single image: `getImage.cgi` will respond with the current image that VC3 is processing, in png format (embedded in the response body)

What follows is the complete documentation of VC3 web service API.

`getDiagnostics.cgi`

This CGI returns a set of diagnostic informations about the application itself and a brief summary of data from the configured probes. Here is an example request:

`http://192.168.1.61/local/VehicleCounter/getDiagnostics.cgi`

and the corresponding response:

```
{
  "id": 15378784,
  "build_version": "v3.3.26",
  "cameraId": "*****",
  "session_id": "*****:20161110:123635",
  "gps_lat": "43.788286",
  "gps_lon": "11.226857",
  "query": "getDiagnostics",
  "localtime": "2016-11-10T17:58:07",
  "platform": "acap",
  "resolution": "320x240",
  "framerate": 20,
  "uptime": 18838,
  "probes": [
    {
      "probeId": 0,
      "state": "idle",
      "count": 1359,
    }
  ]
}
```

```

    "badLength": false,
    "badSpeed": false
  },
  {
    "probeId": 1,
    "state": "idle",
    "count": 508,
    "badLength": false,
    "badSpeed": false
  }
]
}

```

The field "framerate" reports the actual frame rate at which the application is processing the images from the video sensor. The field "uptime" reports the uptime in seconds. The field "probes" reports a brief summary for each probe in use, specifically the probe state (see Tab. 3), the current counter value, and the value of the flags "badLength" and "badSpeed". These flags are set to true if VehicleCounter detected events that may have interfered with the process of estimating the last vehicle's length and speed respectively. This CGI is used by the "VehicleCounter Configuration" page to show general information in the "Real Time Data" section (see Fig. 8a).

```
getProbeState.cgi?pid=X
```

This CGI returns a JSON document with the detailed state of a specific probe X, where X is the probe numerical id, equal to the positional index in the "probes" array provided by the getDiagnostics.cgi. Note that when a new probe is created, its name is set to Probe X to notify the user which id the probe was assigned. Here is an example request:

```
http://192.168.1.61/local/VehicleCounter/getProbeState.cgi?pid=0
```

and the corresponding response:

```

{
  "id": 15378784,
  "build_version": "v3.3.26",
  "cameraId": "*****",
  "session_id": "*****:20161110:123635",
  "gps_lat": "43.788286",
  "gps_lon": "11.226857",
  "query": "getProbeState",
  "localtime": "2016-11-10T18:00:47",
  "probeId": 0,
  "state": "idle",
  "counter": 1363,
  "occupancy": 0.03,
  "signal": 0.75,
  "timesReInit": 1,
  "lastReInitTime": 1478778248,
  "lastVehicleClass": 1,
  "lastVehicleLength": 51,
  "lastVehicleSpeed": 17,
  "fps": 20,
  "lastPassageDate": "2016-11-10",
  "lastPassageTime": "17:59:53",

```

```

    "badLength": false,
    "badSpeed": false
}

```

The response reports, from top to bottom, after the probe id and the probe state (see Tab. 3) the current value of the counter, the occupancy rate, the current value of the signal extracted by the counter sensor, the number of automatic re-initialization occurred, last vehicle's class (see Tab. 4), length (in centimeters) and speed (in Km/h), last passage's date and time and the current value of "badLength" and "badSpeed" flags (they are set to true if inaccuracy in vehicle's length and/or speed estimation has been detected). The pid parameter is mandatory. A request made without the pid parameter or with a non-existent id will result in a code 500 HTTP response (server error). This CGI is used by the "Vehicle-Counter Configuration" page to show specific probe information in the "Real Time Data" section (see Fig. 8b).

```
getCount.cgi?pid=X&interval=Y
```

This CGI returns the number of vehicles counted by one or more probe in a given time interval. The CGI accepts two optional parameters:

- pid=X limits queried data to probe X. If omitted, counter from all configured probes are summed up.
- interval=Y limits queried data to the last Y seconds. If omitted, considers all available data.

Here is an example request:

```
http://192.168.1.61/local/VehicleCounter/getCount.cgi?pid=0&interval=60
```

and the corresponding response:

```

{
  "id": 15378784,
  "build_version": "v3.3.26",
  "cameraId": "*****",
  "session_id": "*****:20161110:123635",
  "gps_lat": "43.788286",
  "gps_lon": "11.226857",
  "query": "getCount",
  "localtime": "2016-11-10T18:02:50",
  "interval": 60,
  "probe": 0,
  "count": 2
}

```

The order of the parameters in the request is non-influential.

```
getRawData.cgi?interval=X&pid=Y&cls=Z&once=
```

This CGI returns a JSON document containing detailed data of each vehicles detected in a given time interval. The CGI accepts several optional parameters:

- `interval=X`. If specified, limits queried data at the last X seconds. If omitted, all available data will be returned. Be cautious, since that could be a significant amount of uncompressed text data.
- `pid=Y`. If specified, limits queried data to probe Y. If omitted, data from all probes will be queried. The value of Y is the valid id of a probe (it usually appear in the probe name), as it is defined in the VC3 configuration (see `getConfiguration.cgi`). If the CGI is given a non-existent probe id, it will return an empty "data" vector.
- `cls=Z`. If specified, limits queried data to vehicle class Z. If omitted, all vehicle's classes will be queried. The value of Z must be in [0; 5], where 0 is the class "unknown", and the numbers from 1 to 5 correspond to vehicle classes as in table 4 (1: motorcycles and small cars; 5: trucks).
- `once=`. If the parameter `once` is defined (there is no need to specify an actual value) it says to VC3 that the query has to be done in "get-once" mode. This means that queried data will be marked so that they will not be repeated in subsequent "get-once" queries. This is particularly useful when you need to transfer data from VC3 into another (usually bigger) database. By using "get-once" queries repeatedly with properly chosen parameters you will ensure that all passages are read from VC3 exactly once, and that no data is lost. Typically, this is done by repeating every T seconds the same query, with a value X of `interval` such that $X > T$.

Here is an example request:

```
http://192.168.1.61/local/VehicleCounter/getRawData.cgi?interval=3600&once=
```

and the corresponding response:

```
{
  "id": 15378784,
  "build_version": "v3.3.26",
  "cameraId": "*****",
  "session_id": "*****:20161110:123635",
  "gps_lat": "43.788286",
  "gps_lon": "11.226857",
  "query": "getRawData",
  "localtime": "2016-11-10T18:04:41",
  "interval": 3600,
  "probe": null,
  "class": null,
  "data": [
    {
      "id": 146368,
      "tm": 1401979539,
      "pid": 0,
      "sid": 0,
      "typ": 0,
      "fos": 5,
      "len": 264,
      "cls": 2,
      "spd": 47,
      "new": 1
    },
  ],
}
```

```

{
  "id": 146369,
  "tm": 1401979548,
  ...

  "spd": 30,
  "new": 1
}
],
"rows": 423
}

```

Each element of the "data" vector correspond to a single vehicle passage, i.e. a record in the VehicleCounter database table. For each passage, the following information are reported:

- "id": passage numerical id;
- "tm": passage timestamp (seconds since epoch)
- "pid": id of the probe that detected the vehicle
- "sid": id of the sensor that detected the vehicle
- "typ": internal use
- "fos": frames spent on the sensor by the vehicle
- "len": estimated vehicle length (in centimeters)
- "cls": vehicle class (see Tab 4)
- "spd": estimated vehicle speed (in Km/h)
- "new": flag for "get-once" queries. The value 1 means this passage has never been queried before, while 0 means this passage has previously been queried by a "get-once" query. Please note that passages extracted by "get-once" queries are updated AFTER they have been returned to the user, so it is correct to have all passages from a "get-once" query with "new" equal to 1.

`getConfiguration.cgi`

This CGI returns the JSON document with the probe's configuration currently in use. It does not accept any parameter, so the sample request is simple:

`http://192.168.1.61/local/VehicleCounter/getConfiguration.cgi`

This is an example of VC3 configuration in JSON format, as returned by the CGI:

```

{
  "id": 15378784,
  "name": "Counting Site",
  "cameraId": "*****",
  "session_id": "*****:20161110:123635",
  "session_start_date": 147877795,
  "session_end_date": 0,

```

```

"gps_lon": "11.226857",
"gps_lat": "43.788286",
"probes": [
  {
    "id": 0,
    "name": "Probe 0",
    "type": "varlen",
    "mask": 0,
    "counter": {
      "id": 0,
      "corners": [
        {
          "x": 0.49690000000000000001,
          "y": 0.58330000000000000004
        },
        {
          "x": 0.49380000000000000002,
          "y": 0.42080000000000000001
        }
      ],
      "upperThreshold": 10,
      "lowerThreshold": 5
    },
    "gates": [
      {
        "id": 0,
        "corners": [
          {
            "x": 0.6655999999999999997,
            "y": 0.4687999999999999999
          },
          {
            "x": 0.16880000000000000001,
            "y": 0.4728999999999999999
          }
        ],
        "delayThreshold": 1,
        "framesThreshold": 1,
        "distance": 500,
        "type": "standard"
      },
      {
        "id": 1,
        "corners": [
          {
            "x": 0.66410000000000000002,
            "y": 0.5292
          },
          {
            "x": 0.17030000000000000001,
            "y": 0.5333
          }
        ],
        "delayThreshold": 1,
        "framesThreshold": 1,
        "distance": 500,
        "type": "standard"
      }
    ]
  },
  {
    "id": 1,
    "name": "Probe 1",
    "type": "varlen",

```


is the new configuration in JSON format (see `getConfiguration.cgi`).


```
getPassages.cgi?res=X&pid=Y&cls=Z
```

This CGI returns the passages stored in the database, aggregated to meet the given temporal resolution. The CGI accepts the following parameters:

- `res=X`. Specifies the resolution in seconds of the returned data. That is, queried data are divided in "bins" that correspond to time interval of width equal to X seconds, so that `res=60` provides statistics with the resolution of a minute, `res=3600` provides statistics with the resolution of an hour and so on. If omitted, defaults to 3600.
- `pid=Y`. If specified, limits queried data to probe Y. If omitted, data from all probes will be queried. The value of Y is the valid id of a probe (it usually appear in the probe name), as it is defined in the VC3 configuration (see `getConfiguration.cgi`). If the CGI is given a non-existent probe id, it will return an empty "data" vector.
- `cls=Z`. If specified, limits queried data to vehicle class Z. If omitted, all vehicle's classes will be queried. The value of Z must be in [0; 5], where 0 is the class "unknown", and the numbers from 1 to 5 correspond to vehicle classes as in table 4 (1: motorcycles and small cars; 5: trucks).

Here is an example request:

```
http://192.168.1.61/local/VehicleCounter/getPassages.cgi?res=60
```

and a sample response:

```
{
  "id": 15378784,
  "build_version": "v3.3.26",
  "cameraId": "*****",
  "session_id": "*****:20161110:123635",
  "gps_lat": "43.788286",
  "gps_lon": "11.226857",
  "query": "getPassages",
  "localtime": "2016-11-10T18:08:54",
  "probe": null,
  "class": null,
  "resolution": 60,
  "data": [
    {
      "bin": 23828022,
      "count()": 15,
      "avg(spdx)": 20.8,
      "avg(len)": 274.866666666667
    },
    {
      "bin": 23828023,
      "count()": 4,
      "avg(spdx)": 11.75,
      "avg(len)": 259.75
    },
    {
      "bin": 23828024,
      "count()": 10,
      "avg(spdx)": 26.1,
      "avg(len)": 261.4
    }
  ],
}
```

```

...
{
  "bin": 23828493,
  "count()": 8,
  "avg(sp)": 27.625,
  "avg(len)": 335.375
}
]
}

```

Each element of the "data" vector correspond to a single "bin", i.e. to a single time interval of width specified by the value of "resolution" (expressed in seconds). Each elements contains the following data:

- "bin": id of the current time interval. This number is the most significant part of the "tm" (time) value of all passages that have been aggregated in this bin. In other words, each bin collects the data of passages for which $tm/resolution = bin$, or equivalently, that have the value of "tm" in the interval $[bin \times resolution; (bin + 1) \times resolution)$.
- "count()": the number of vehicles detected in this time interval.
- "avg(sp)": the average speed of vehicles in this time interval, in Km/h.
- "avg(len)": the average length of vehicles in this time interval, in centimeters.

`getStatsPerClass.cgi?interval=X&pid=Y` (experimental)

This CGI returns some statistics aggregating data so to keep separated the vehicles from the different classes. The CGI accepts the following parameters:

- `interval=X`. If specified, limits queried data at the last X seconds. If omitted, all available data will be queried.
- `pid=Y`. If specified, limits queried data to probe Y. If omitted, data from all probes will be queried. The value of Y is the valid id of a probe (it usually appear in the probe name), as it is defined in the VC3 configuration (see `getConfiguration.cgi`).
If the CGI is given a non-existent probe id, it will return an empty "data" vector.

Here is an example request:

`http://192.168.1.61/local/VehicleCounter/getStatsPerClass.cgi`

and a sample response:

```

{
  "id": 15378784,
  "build_version": "v3.3.26",
  "cameraId": "*****",
  "session_id": "*****:20161110:123635",
  "gps_lat": "43.788286",
  "gps_lon": "11.226857",
  "query": "getStatsPerClass",

```

```

"localtime": "2016-11-10T18:10:01",
"interval": null,
"probe": null,
"count": [
  {
    "cls": 0,
    "count()": 1376,
    "min(len)": 0,
    "avg(len)": 558.678052325581,
    "max(len)": 21365,
    "min(spд)": 0,
    "avg(spд)": 55.452761627907,
    "max(spд)": 169
  },
  {
    "cls": 1,
    "count()": 8181,
    "min(len)": 6,
    "avg(len)": 169.316831683168,
    "max(len)": 250,
    "min(spд)": 0,
    "avg(spд)": 36.6078718983009,
    "max(spд)": 169
  },
  {
    "cls": 2,
    "count()": 11254,
    "min(len)": 251,
    "avg(len)": 381.793051359517,
    "max(len)": 450,
    "min(spд)": 2,
    "avg(spд)": 38.3273502754576,
    "max(spд)": 161
  },
  {
    "cls": 3,
    "count()": 11428,
    "min(len)": 451,
    "avg(len)": 509.028438921946,
    "max(len)": 600,
    "min(spд)": 4,
    "avg(spд)": 40.4398844942247,
    "max(spд)": 169
  },
  {
    "cls": 4,
    "count()": 3085,
    "min(len)": 601,
    "avg(len)": 673.754943273906,
    "max(len)": 800,
    "min(spд)": 6,
    "avg(spд)": 48.3896272285251,
    "max(spд)": 169
  },
  {
    "cls": 5,
    "count()": 1091,
    "min(len)": 801,
    "avg(len)": 1101.03116406966,
    "max(len)": 13080,
    "min(spд)": 8,
    "avg(spд)": 55.0760769935839,
    "max(spд)": 169
  }
]

```

```
]
}
```

Each element of the "data" vector correspond to a single "bin", i.e. to a single vehicle class. Each elements contains the following data:

- "cls": id of the current vehicle class. The class 0 is the class unknown, while classes from 1 to 5 correspond to classes in table 4.
- "count()": the number of vehicles detected in this class.
- "min(len)": the minimum length of vehicles in this class, in centimeters.
- "avg(len)": the average length of vehicles in this class, in centimeters.
- "max(len)": the maximum length of vehicles in this class, in centimeters.
- "min(sp)": the minimum speed of vehicles in this class, in Km/h.
- "avg(sp)": the average speed of vehicles in this class, in Km/h.
- "max(sp)": the maximum speed of vehicles in this class, in Km/h.

```
getSpeedDistribution.cgi?res=X&pid=Y&cls=Z (experimental)
```

This CGI returns the distribution of estimated speed with the given resolution, eventually limited to the specified probe and vehicle's class. The CGI accepts the following parameters:

- res=X. Specifies the resolution at which the data will be aggregated. X is expressed in Km/h, so res=5 means that queried data will be aggregated in bins of 5 Km/h of width. If omitted, it defaults to 10 Km/h.
- pid=Y. If specified, limits queried data to probe Y. If omitted, data from all probes will be queried. The value of Y is the valid id of a probe (it usually appear in the probe name), as it is defined in the VC3 configuration (see `getConfiguration.cgi`). If the CGI is given a non-existent probe id, it will return an empty "data" vector.
- cls=Z. If specified, limits queried data to vehicle class Z. If omitted, all vehicle's classes will be queried. The value of Z must be in [0; 5], where 0 is the class "unknown", and the numbers from 1 to 5 correspond to vehicle classes as in table 4 (1: motorcycles and small cars; 5: trucks).

Here is an example request:

```
http://192.168.1.61/local/VehicleCounter/getSpeedDistribution.cgi
```

and a sample response:

```
{
  "id": 15378784,
  "build_version": "v3.3.26",
  "cameraId": "*****",
  "session_id": "*****:20161110:123635",
```

```

"gps_lat": "43.788286",
"gps_lon": "11.226857",
"query": "getSpeedDistribution",
"localtime": "2016-11-10T18:10:54",
"probe": null,
"class": null,
"resolution": 10,
"data": [
  {
    "bin": 0,
    "count()": 2195
  },
  {
    "bin": 1,
    "count()": 6272
  },
  {
    "bin": 2,
    "count()": 7981
  },
  ...
  {
    "bin": 89,
    "count()": 1
  },
  {
    "bin": 100,
    "count()": 1
  },
  {
    "bin": 105,
    "count()": 1
  }
]
}

```

Each element of the "data" vector correspond to a single "bin" and contains the following data:

- "bin": id of the current bin. Each bin collects the data of passages for which $\text{spd}/\text{resolution} = \text{bin}$, or equivalently, that have the value of "spd" in the interval $[\text{bin} \times \text{resolution}; (\text{bin} + 1) \times \text{resolution})$.
- "count()": the number of vehicles detected in this speed interval.

Please note that not all bins are necessarily present in the "data" vector: only bins with non-zero "count()" value are reported.

`getLengthDistribution.cgi?res=X&pid=Y&cls=Z` (experimental)

This CGI returns the distribution of estimated vehicles length with the given resolution, eventually limited to the specified probe and vehicle's class. The CGI accepts the following parameters:

- `res=X`. Specifies the resolution at which the data will be aggregated. `X` is expressed in cm, so `res=5` means that queried data will be aggregated in bins of 5 cm of width. If omitted, it defaults to 10 cm.
- `pid=Y`. If specified, limits queried data to probe `Y`. If omitted, data from all probes will be queried. The value of `Y` is the valid id of a probe (it usually appear in the probe name), as it is defined in the VC3 configuration (see `getConfiguration.cgi`). If the CGI is given a non-existent probe id, it will return an empty "data" vector.
- `cls=Z`. If specified, limits queried data to vehicle class `Z`. If omitted, all vehicle's classes will be queried. The value of `Z` must be in `[0; 5]`, where 0 is the class "unknown", and the numbers from 1 to 5 correspond to vehicle classes as in table 4 (1: motorcycles and small cars; 5: trucks).

Here is an example request:

`http://192.168.1.61/local/VehicleCounter/getLengthDistribution.cgi`

and a sample response:

```
{
  "id": 15378784,
  "build_version": "v3.3.26",
  "cameraId": "*****",
  "session_id": "*****:20161110:123635",
  "gps_lat": "43.788286",
  "gps_lon": "11.226857",
  "query": "getLengthDistribution",
  "localtime": "2016-11-10T18:11:55",
  "probe": null,
  "class": null,
  "resolution": 10,
  "data": [
    {
      "bin": 0,
      "count()": 92
    },
    {
      "bin": 1,
      "count()": 285
    },
    {
      "bin": 2,
      "count()": 517
    },
    ...
    {
      "bin": 851,
      "count()": 1
    },
    {
      "bin": 860,
      "count()": 1
    },
    {
      "bin": 868,
      "count()": 1
    }
  ]
}
```

}

Each element of the "data" vector correspond to a single "bin" and contains the following data:

- "bin": id of the current bin. Each bin collects the data of passages for which $len/resolution = bin$, or equivalently, that have the value of "len" in the interval $[bin \times resolution; (bin + 1) \times resolution)$.
- "count()": the number of vehicles detected in this length interval.

Please note that not all bins are necessarily present in the "data" vector: only bins with non-zero "count()" value are reported.

`getBuildVersion.cgi`

This CGI returns the application build version. No parameter is required.

Here is an example request:

`http://192.168.1.61/local/VehicleCounter/getBuildVersion.cgi`

and the corresponding response (pure text, not JSON):

v3.3.26

`cleanDB.cgi`

This CGI commands VehicleCounter to clean the internal SQLite3 database. When the application receives this request, if no error occurs it immediately deletes all the passages stored in the database so far, without requesting any confirmation, so be cautious in using this CGI as there is no way to recover the data once they have been deleted.

`http://192.168.1.61/local/VehicleCounter/cleanDB.cgi`

If no error occurs, the response is "200 OK".