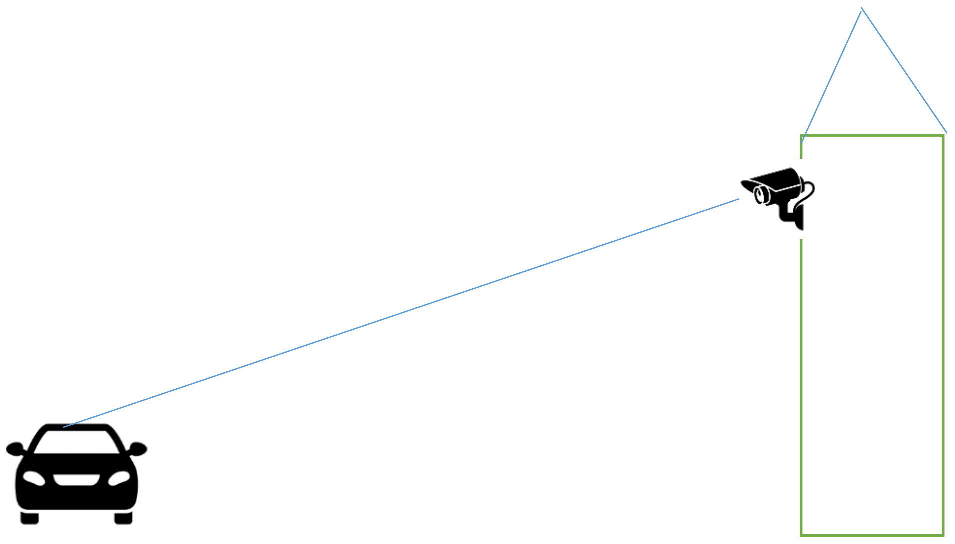
Vehicle Seed Detector

1. Summary

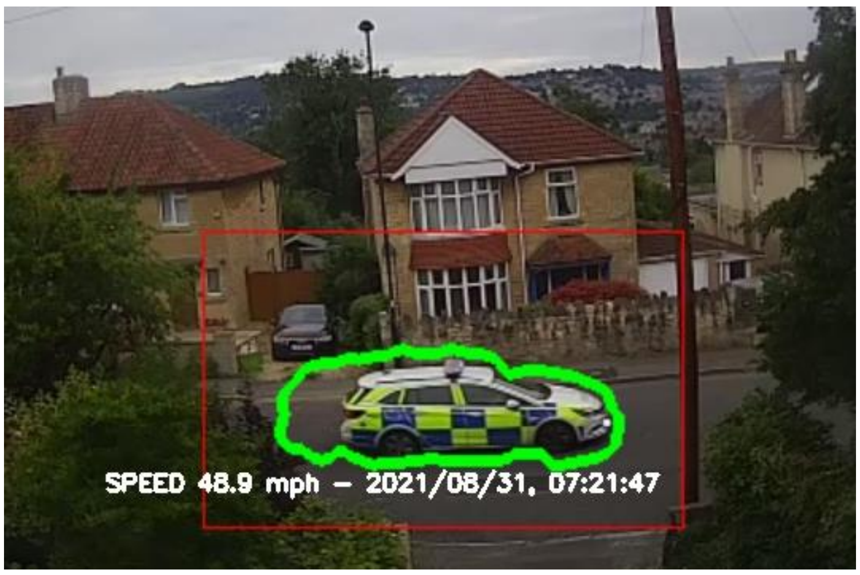
The home_speed_detection set of python scripts is designed to log vehicle count and speeds using a camera and image recognition to track vehicles. The resultant information can be used to inform local safety groups, councils and law enforcement of particular traffic issues, or to log traffic flows when some roadworks, diversions or permanent re-routing takes place.

Based on work originally by Claude Pageau , https://github.com/pageauc/rpi-speed-camera/tree/master/, Ronit Sinha, https://github.com/ronitsinha/speed-detector, and many others including of course the OpenCV project and team.

The program works by capturing images on a camera perpendicular to, at a distance from and at an elevation to, a single or dual track road.



The field of view, shown as the red bounding rectangle is a known size in pixels, and a calibration is performed to relate pixel spacing to actual length dimensions. Then by tracking the object horizontally, recording the pixel shift distance and the time taken, object speed can be calculated.



2. Details

Image recognition

The heart of the program is the image recognition and tracking.

Image recognition is performed using OpenCV API in Python. Image frames are captured from the image source, together with a timestamp for each frame. The image is first cropped to a region of interest (the red rectangle). The cropped background image provides the base, then the next image is compared and the differences created as a new image. OpenCV's contour recognition, with some filtering, is used to find the largest contour in the difference image. Checks are performed to make sure that the detected area is likely to be a vehicle, by specifying a minimum area that a contour has to exceed. The contour, together with its properties such as position and the image timestamp are used to create a Vehicle object.

Each subsequent frame image is examined in the same way, and if they prove consistent in terms of size and tracking position, a tracking record is created for each and added to a list until sufficient records have been created to arrive at a reasonable average speed for the object.

Direction determination

We need to be able to determine direction of travel for each contour, this is designated Left To Right (L2R) or Right To Left (R2L), referencing the screen display. When we have two or more samples, we can do this by simply checking for increasing (L2R) or decreasing (R2L) values of x. But for the first sample we can only know the direction of L2R, since as it enters the FoV, x will always be zero. If x is close to the FoV width, then probably the direction is R2L. Establishing the direction early is important since we need to check each sample for x-position relative to the last. If the sample suddenly apparently reverses direction there is assumed to be something wrong with it, or it belongs to another contour.

Position tracking

The system uses the x-position of the contour to provide the distance difference for speed calculation. But where is x?

We could use centroid tracking, as some systems do, but this relies on always having the whole contour in the field of view all the time. Otherwise as the contour enters or leaves the FoV, the contour size changes, and the centroid x position moves as a result of area as well as position. It can even move backwards! So we choose to use leading edge positioning. In this case we look for the first edge of the contour as it enters the FoV, then track this as it moves across, a bit like an electric eye detector. It turns out that this is fine for contours travelling Right to Left since the x-position from OpenCV is the bottom left-hand corner and progresses in decreasing x across the screen.

For Left to Right however, on entering the FoV, the x position is always zero, at least until the whole contour is in view. so we now have trailing edge detection.

To solve this we use the contour aspect of width and add it to the x-position when the contour is travelling Left to Right, thus restoring the leading edge and we can use the same code for each direction.

Timing

Vital to the accuracy of the speed calculation is the timing between track events. The chosen method is to use OpenCV's video frame time together with the local time to get an absolute timestamp for each frame. It is also important to use a small frame buffer, formed from a Queue, so that processing delays do not affect the timestamp value. Note that in the case of a file source, the file start time is used as the base, so that extracted images and statistics reflect the time of recording, not the time of replay.

Average speed calculation and accuracy.

Each Vehicle has a list of tracking records, to a maximum as set by the user. Individual speed records are created for each entry by determining the change in x-position between records, and the difference in timestamp between each record, then using the speed = distance/time equation to obtain the speed for each frame.

The contour from the image is subject to many variables, such as lighting, reflections, passing vehicles and so on, so any one speed calculation may differ from the next due to 'jitter' in the x position. Taking a number of samples allows a mean speed to be calculated, together with a standard deviation. This smooths out the jitter, as well as providing a figure of merit for the reliability of the calculation.

Of course there is no speed information for the first sample, since there is no difference information, so this is excluded from the mean calculation.

As always there is a trade-off between the number of samples, which should notionally be as high as possible, and other factors such as vehicle size, field of view size, max speed detectable and others.

The reason for this is that the number of samples may cause the x-positon to exceed the width of the field of view in the time taken for the vehicle to pass. As an example, good results can be obtained from a 30fps camera, with a 200 pixel wide field of view at a 30 m distance if the sample size is set to about 5 or 6. Trial and error is needed here to optimise.

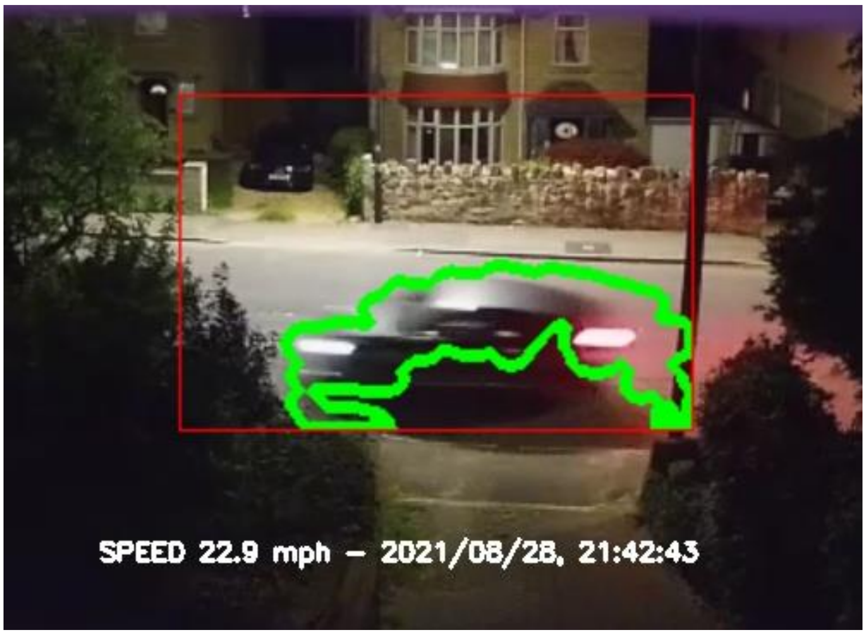
Sources

The program accepts a network IP camera using e.g rtsp protocol, the Raspberry PI camera or a pre-recorded file.

Network cameras are probably best if the camera positioning is to be out in the weather. Mounting and positioning the camera so it does not move is important to prevent camera shake, which would create its own contours. Perhaps counter-intuitively, high resolution is not required, rather a low resolution stream is preferred since it gives the program fewer calculations to do. In general 640 x 480 gives perfectly good results.

It also doesn't need to be colour, unless you want your recorded images to be so. Almost the first thing image recognition programs do is to convert to monochrome, since there is little useful motion information in colour images. If you are going to use in low-light conditions, this should also be a benefit.

A word on night-time operation. A well-lit street with a good quality monochrome camera and maybe some contrast tweaking will still give good results.



Unlit roads probably won't work, since there is headlight and taillight glare and obtaining a complete contour is very difficult. You could dedicate a low-light camera in these circumstances, or if you can programmatically control your camera, adjust the exposure, contrast etc according to the prevailing light conditions.

Raspberry PI operation is supported, good daytime results can be obtained by viewing through a window, else you will need a weather-proof housing for your RPI with camera. But it makes a good portable capture system, self-contained and needing only power.

The RPI camera in general will not perform as well as a dedicated IP camera, in particular we need good and consistent framerates, and this varies considerably (and deliberately) with the resolution chosen for the PiCam. For high-integrity results, we need to operate at better than 20fps consistently, and this is a stretch for the RPI and camera at, say 640x480 resolution. The RPI4B just about manages it. But the camera is more directly controllable than an IP camera, so it's easier to adjust for different lighting and exposure conditions.

The lens system on the RPI is also limited, so quality is less good. This is apparent in the constrained perspective, necessitating careful calibration in the different direction channels. Practically speaking, the RPI system is probably only good up to about 30mph detection.

Queue buffering and threading is used to smooth the processing load, although Python's GIL realy constrains the value of threading. Remember too that the system needs some horse-power, so a RPI4B with 2G of RAM is probably a minimum.

File sourcing is very useful for debugging, or trying out potential locations without setting up the entire system. Simply record a video of the target road for a while, then use this as a source for analysis. Obviously this is not practical for protracted monitoring, since the resultant video file quickly gets huge, but up to an hour should be feasible. As above, make sure that the video is free from shake, you probably can't hold a camera phone still in your hand for too long so use a mount.

Field of View selection

The program supports dynamic selection and changing the FoV. With the program running, type ‘m’ (for mask) and a Region of Interest window will open. Use the cursor to place a rectangle and drag the corner to the appropriate position. Clicking opens a dialog to save the new FoV or discard. The new FoV is available immediately, and also stored in an overlay-specific json file, so when you restart the program with the same overlay, the FoV is restored. For headless operation, you can edit the json file directly, in this case the new value is not used until the program is restarted.

Calibration

In order to convert pixels to distance you must supply a conversion in the ini file. To help this, the main image (assuming a GUI implementation) shows a 10 pixel horizontal grid. Choose a couple of fixed markers, perhaps gateposts or markings on the road and measure the actual distance (in mm) then find the markers on the image and count the pixels between them .

You can provide different values for L2R and R2L, since perspective makes objects further away appear smaller. Ad these values in your ini file for your overlay, or in the main config.ini if not using overlays.

Then conduct a live test. This can be by driving past the camera at a known speed (use satnav not the vehicle speedometer which is always calibrated high). Or compare with a radar gun or similar.

You can record the live image from your camera so you have a permanent calibration file for any future tweaks. This is invoked by the ‘calibrate’ flag in the ini file and creates a ‘calibrate.mp4’ file in the current working directory. Don’t forget to turn it off!.

It is likely that your first calibration will yield results that are too slow. This is because the contour envelope is always bigger than the actual vehicle. so tweak the numbers until you get the right result.

Data Logging

The main output of the program is the data file(s) produced detailing the records of each detection event.

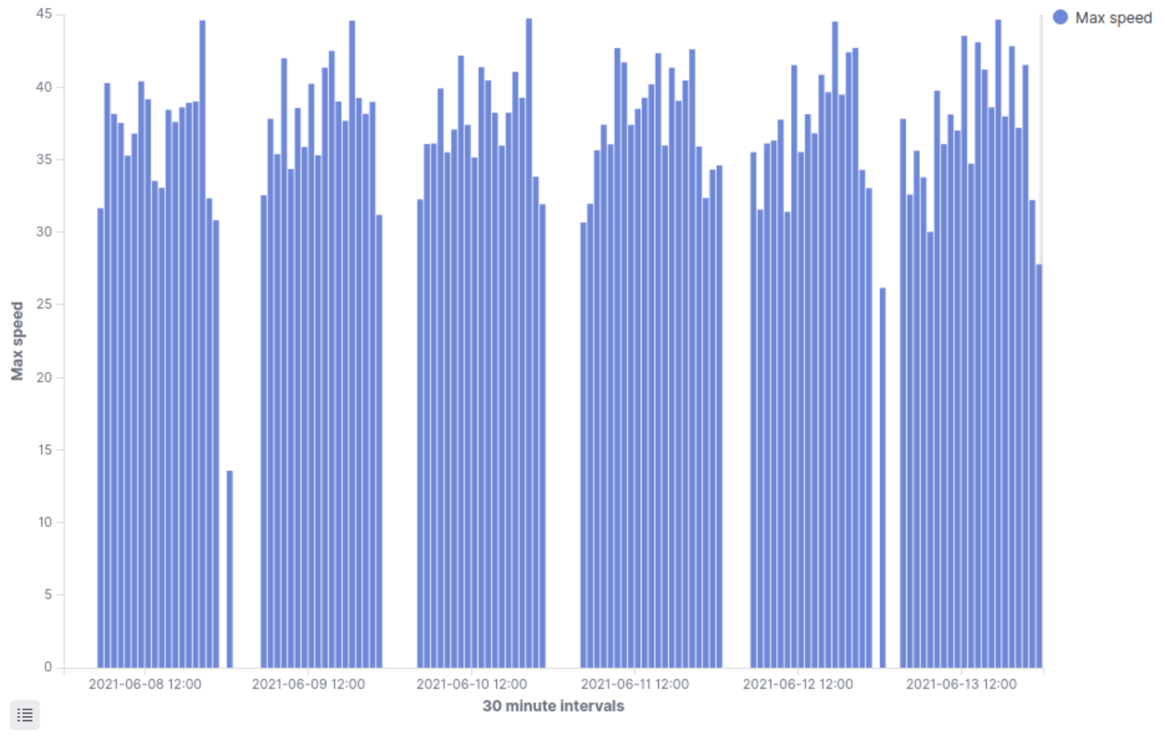
The simplest rendition of this is in CSV format, a plain-text comma delimited file, with one record per line. This can be ingested into many different data visualisers, databases or just a spreadsheet.

The record format for each completed detection event is:
“time stamp (year-month-day hour:minute:second)”,
average speed,
”speed unit”,
speed standard deviation,
“filename of the associated image”,
detected object size,
”direction of travel”,
“camera location”

Items in quotes (“”) are strings, others are number values.

Not tested in this version is also a Sqlite database entry option, which may or may not work.

Visualisation and reporting is probably the most useful thing to do with this data, here is a screenshot of a graph produced using Elasticsearch and Kibana showing the maximum daytime speeds in 30 minute slots recorded each day over a 5 day period.



3. USER SETTINGS VIA THE INI FILE.

The config.ini stores user settable parameters that are loaded on program start. A subset of these can be used in an overlay ini file that overwrites just those entries. So usually you do not need to provide all values in your specific overlay file.

The ini file structure is [category] followed by key value pairs. The overlay file needs the category, as well as a particular key and value.

Just use plain text. Do not use quotes around string values e.g file names and paths, or include extra (escaped) backslashes etc.

The easiest way to make an overlay ini file is to copy the config.ini with a new filename, then delete all the entries you do not want to change. Just edit the remaining entries with your preferred values.

Keys must be exactly as shown below, invalid values will mostly be ignored or defaults used. But you can probably still break the program with silly values.

In the following list, the categories are in square brackets, followed by one or more key= value entries.

Stream source settings

These determine where the program gets its data from. Sources can be file or webcam or pi camera.

If you enable file as a source you should supply at least the filename, assumed to be in the same directory as the scripts are run from, or the fully qualified file pathname.

For camera sources ensure *file_src* is *False* then select type of camera in WebCam settings

[Source]

file_src=False - boolean value determines whether the source is a file

source_file_name = - the name of the mp4 file to be loaded, with extension.

file_loop = *True* - boolean value if true plays file continuously otherwise stops at the end of file.

Calibration settings

Includes calibration mode and the values that translate pixels to millimetres

[Calibration]

calibrate = *False* Boolean value if true creates a calibration image file with calibration hash markers 10 px per mark. This will also save a calibration video file of the scene as viewed by the camera. You can use this to tweak calibration values without having to run an actual test every time.

cal_obj_px_L2R = *80* integer value only, the Left to Right Length of your calibration object in pixels as seen on the calibration image.

cal_obj_mm_L2R = *4700.0* floating point value, Left to Right2R length of the calibration object in millimetres

cal_obj_px_R2L = *85* integer value only, Right to Left length of a calibration object in pixels

cal_obj_mm_R2L = *4700.0* floating point value, Right to Left length of the calibration object in millimetres

These are best set in an overlay file, since they will change with different cameras, locations etc.

# Note if tested speed is too low increase appropriate *cal_obj_mm* value and redo speed test for desired direction.

hash_colour = *255,0,0* the colour of the hash marks on the image in RGB format

motion_win_colour = *0,0,255* the colour of the field of view rectangle

Overlay Settings

When overlays are enabled, edit settings in specified overlay file located in overlays folder.

Settings from config.ini will be used if not specified in the overlay file

[Overlays]

overlayEnable = *True* Boolean value, if true, enables the use of overlays.

overlayName = *name* - a string value naming the overlay file stored in the overlays subfolder. Don't include the .ini extension

GUI Settings

Displays opencv windows on gui desktop. Turn this off for pure console mode, or to save processing resources.

[GUI]

gui_window_on = *True*   Boolean value. If true turn on all desktop GUI openCV windows. False=Don't show (turn off for SSH operation) .

gui_show_camera = *True* Boolean value. If true show the camera view, otherwise hide but show other windows if enabled

show_thresh_on = *False* Boolean value. if True, display the (cropped) threshold window that shows the contour detected. Only use if you need to make fine adjustments to camera settings for contrast, brightness etc. and want to see how well the object is being picked out.

show_crop_on = *False*   Boolean value. If True shows the field of view in a separate window.

Logging Settings

Sets the conditions for application logging, to file and/or to console.

There are two log files active at any time. The first is the application log that keeps track of the overall application, mostly on start and stop but also catches application errors.

The second is specific to the active overlay, and relates all the actions while an overlay is active. This file can also be limited in size and rolled over. See below.

Log file format is largely set in the application, but there is also a logging.conf file that formats the application log only. To change the log file name and/or path, set the args value of the handler_appfileHandler section to the new path/name. See Python Logging module for details.

[Logging] #Display and Log settings

verbose = *True* Boolean value. If True displays all the information available including debug.

display_fps = *False*   Boolean value. If True , log the average frame rate every 300 frames

log_data_to_CSV = *True* Boolean value. If True, save the speed data as CSV comma separated values. The file is named after the overlay and stored in the *Data* directory.

log_data_to_DB = *False* Boolean value, If True save the speed data data to SQL database. This is an untested feature so should be False.

loggingToFile = *True* Boolean value. If True, save logging to file. The file is named according to the current overlay and stored in the *Overlays* directory

maxLogSize= *1E6* an integer value representing the maximum overlay log file size (in bytes) before log rolls over. Set to 0 disables this, the file will grow forever.

logBackups = *1* an integer that specifies the number of overlay log copies to hold. When the log rolls over, the previous log will be suffixed 1,2 etc. Needs to be at least 1 for the *maxLogSize* to be effective. 0 disables.

Motion and Detection Settings

[Motion]

SPEED_MPH = *True* Boolean value to set speed units. If True speed units are mph, if False kph

track_counter = *5* integer value to set the number of speed measurements to take for averaging and to trigger a speed photo. Default= 5.

This number should be adjusted dependant on the width of the window, the size of the vehicle that can be detected and the maximum speed detectable. If there is a wide detection window, this number can be increased. . See the section on Speed Averaging and Accuracy.

MIN_AREA = 1000 Integer value(in square pixels) that sets the minimum area of a detected object that will be counted. Set this to a number large enough to catch the smallest vehicle you want to detect, but large enough to eliminate false contours like waving bushes or people walking past. Depends on the distance from the target, resolution and so on. some trial and error may be appropriate. The verbose log shows the contour area detected for each detection to give you some idea.

show_out_range = False Boolean value. If True, 'missed' detections will be shown in the log. Gives some idea of the detection accuracy while setting up.

x_diff_max = 40 Integer value in pixels. Default= 40 . Motion events will be discard if any detection x-shift exceeds this. Provides some degree of filtering for multiple targets or difficult lighting conditions that lead to indistinct contours.

x_diff_min = 1 Integer value in pixels. Default= 1 Excludes motion events x-shift is less than this, which may indicate a stationary vehicle or just noisy image.

y_diff_max = 10 Integer value in pixels. Discard motion events if there is a sudden shift in y-position greater than this.

track_timeout = 1 floating point value in seconds. If greater than 0, frame processing is skipped for this time after a motion event has completed. Provides some protection against counting the same object multiple times.

max_speed_over = 8 floating point number in speed units. Motion event speeds will be discarded if they do not reach this threshold. Setting to 0 will record all events. Can be useful to exclude pedestrians and/or bikes etc., or track only fast objects.

max_speed_count=65 floating point value in speed units. Do not count anything over this speed since it is probably wrong. Used to keep the data files (DB or CSV) clean of spurious readings.

Note: To see motion tracking crop area on images, Set variable image_show_motion_area = True

The field of view should be set dynamically on the image, but these values will be used If stored values are not found.

x_left = 220 integer value in pixels indicating left side of FoV rectangle.

x_right = 430 integer value in pixels indicating right side of FoV rectangle.

y_upper = 20 integer value in pixels indicating top side of FoV rectangle.

y_lower = 160 integer value in pixels indicating bottom side of FoV rectangle.

Camera Settings

This section controls the camera settings. Two types are supported, IP webcam and the Raspberry Pi camera. Webcams should be accessed using the rtsp protocol or as a USB device. See Sources discussion above.

[WebCam]

CAM_LOCATION = *Location1* String value that enables you to name your camera source for use in the DB/CSV data file.

WEBCAM = *True* Boolean value. If True the source is a Webcam, USB or IP, False indicates that the camera is Raspberry PI. (But this is overridden if not on a RPI system.)

WEBCAM_SRC = *rtsp://192.168.1.1:554/11* string value which can be eg 0 for a USB camera device connection number or RTSP cam string eg "rtsp://192.168.1.101/RtspTranslator.12/camera". You will need to find your camera settings for this. Check that you can receive images separately from your installed camera before attaching to this program.

Camera image resolution – check the *actual* resolution your stream provides. This may not be as advertised.

WEBCAM_WIDTH = *640* Integer value in pixels. USB Webcam Image width ignored for RTSP cam

WEBCAM_HEIGHT = *480* Integer value in pixels USB Webcam Image height ignored for RTSP cam

WEBCAM_HFLIP = False Boolean value. If True the image will be horizontally flipped. Avoid if possible (by inverting the camera physically) since this sucks resources.

WEBCAM_VFLIP = False Boolean value. If True the image will be vertically flipped. Avoid if possible (by inverting the camera physically) .

[PiCamera]

CAMERA_WIDTH = *320* Integer value in pixels, get from your PiCamera specs Default=320

CAMERA_HEIGHT = *240* Integer value in pixels, get from your PiCamera spec

CAMERA_FRAMERATE = *20* Integer value in frames per second. Default= 20 get from your PiCamera specs

CAMERA_ROTATION = *0* Rotate camera image, valid values are 0, 90, 180, 270

CAMERA_VFLIP = *True* Boolean value, If True flip the camera image vertically

CAMERA_HFLIP = *True* Boolean value. If True flip the camera image horizontally

Note as for the webcam, try to avoid unnecessary image manipulation by careful camera positioning.

Stored Image Settings

[Image]

image_path = *media/images* string value of folder name to store images

image_prefix = *speed_* string value of a prefix for the image name

image_format = *.jpg* string value of extension only jpg is supported.

image_show_motion_area = *True* Boolean value If True, display field of view rectangle area on saved images

image_filename_speed = *False* Boolean Value. If True, prefix the filename with speed value

image_text_on = *True* Boolean value. If True show text on speed images False= No Text on images

image_text_bottom = *True* Boolean Value. If True show image text at bottom otherwise at top *image_font_size* = *12* Integer value in pixels. Font text height for text on images

image_font_scale =*0.5* Floating point value. Font scale factor that is multiplied by the font-specific base size.

image_font_thickness = *2* integer value in pixels. Font text thickness for text on images

image_font_color = *(255, 255, 255)* Tuple value in RGB form representing the font colour. Default= (255, 255, 255) White

image_bigger = *3.0* Floating point value to resize saved speed image by specified multiplier value. Default= 3.0 min=0.1

image_max_files = *0* integer value of the maximum number of files to maintain, then oldest are deleted Default=0 (off)

Optional: Save Most Recent files in recent folder

imageRecentMax = 0 integer value 0=off, >0 will maintain specified number of most recent files in *imageRecentDir* as symlinks

imageRecentDir = *media/recent* string value of path for the recent files. Default= "media/recent"

Manage Free Disk Space Settings

The overall disk space used can be limited using these settings.

[Files]

spaceTimerHrs = *0* floating point value to specify the interval between disk space checks. Default= 0 (no check)

spaceFreeMB = *500* floating point value of target free space in MB required. Default= 500.

spaceMediaDir = *media/images* string value of path where directory walk for free space will start Default= 'media/images'.

spaceFileExt = *jpg* string value of extension of files to search when deleting oldest files. only jpg is supported.

Optional Manage Subdirectory creation by time, number of files or both (not recommended)

imageSubDirMaxFiles = *2000* integer value. If exceeded a new subdirectory will be created. 0=off

imageSubDirMaxHours = *0* floating point value if greater than 0 creates a new dated sub-folder if value exceeded

[Sqlite3]# not tested in this version

DB_DIR = *data* string value for the path to store the database file

DB_NAME = *speed_cam.db* string value name of the database to be used/created

DB_TABLE = *speed* string value for the name of the table to be created/used within the database.