

Video Speed Tracker

Paul Reynolds

reynolds@virginia.edu

www.cs.virginia.edu/~pfr

<https://github.com/pfr/VideoSpeedTracker>

Feb 2016

Overview

The Video Speed Tracker (VST) is an open source, technically sound, vehicle speed measuring system that can track bidirectional traffic, one lane in each direction (e.g. a typical residential street). Software for VST can be found at <https://github.com/pfr/VideoSpeedTracker>. Beyond a typical home computer, VST requires video from an HD camera, many of which can be purchased for less than \$100 (e.g. the Foscam Fi9103EP, power over Ethernet, outdoor camera). If you plan to modify and recompile VST then you need to have OpenCV c 2.4.11 installed.

The VST software package comprises two components: 1) the **Video Speed Tracker** and 2) a **final highlights video processor**. The Video Speed Tracker (VST) takes user provided setup information and inputs either a single HD video or a directory of HD videos, containing recording(s) of passing traffic. VST outputs:

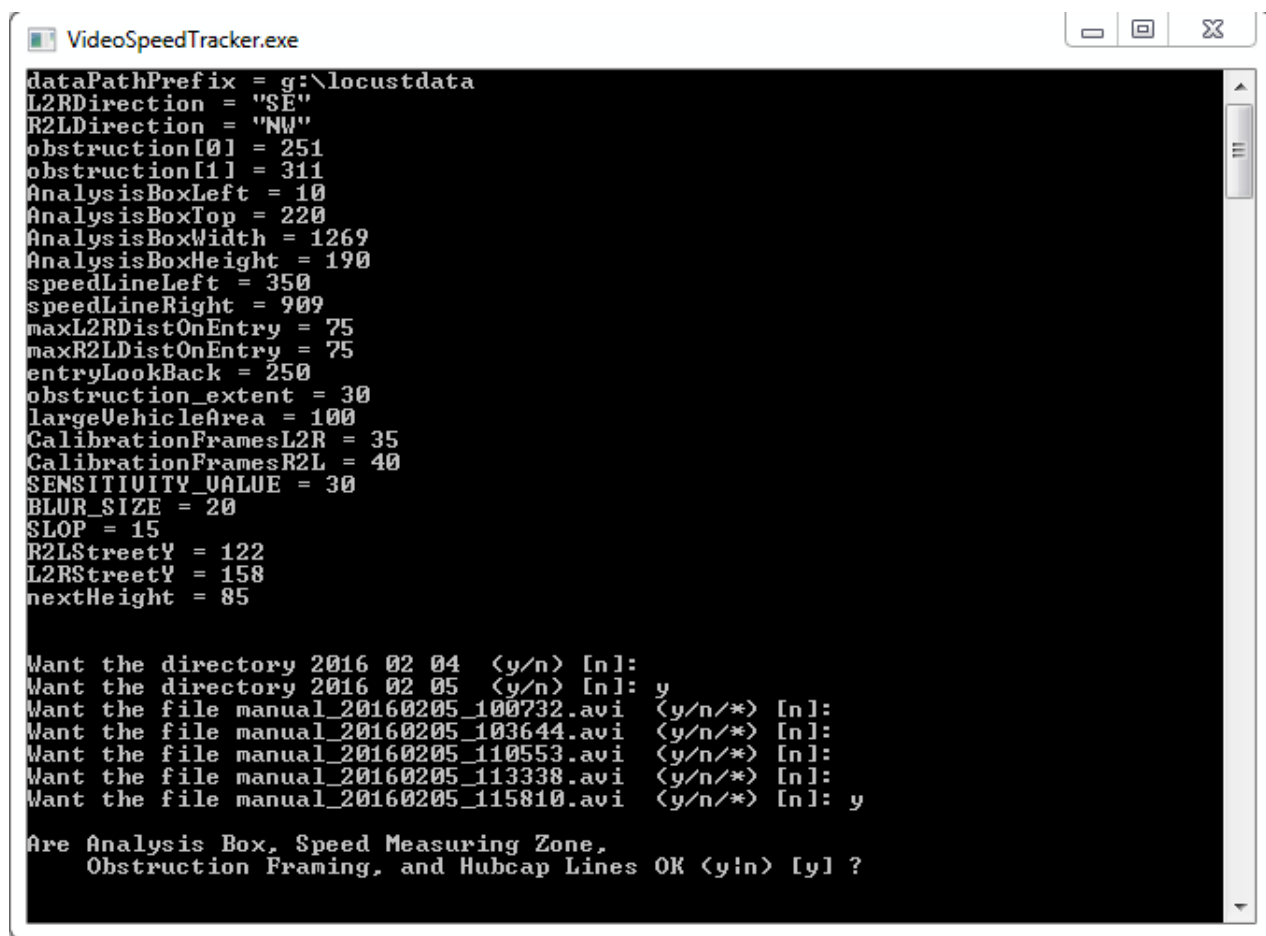
1. A csv (comma separated) file with one entry for each vehicle tracked in the video(s). Each vehicle entry includes video file name, start and end frame numbers the vehicle was tracked, its direction, its profile area and an estimated speed. Profile area can be handy for separating buses and large trucks from other vehicles. Estimated speed entries support speed data analysis of any sort imaginable.
2. A trace (debug) file, if you request it. This file contains copious information for determining how the tracker derived a final speed for a given vehicle.
3. A highlights video, if requested, where the user can select a speed range for identifying vehicles to be captured in the highlights video. Vehicle area is also used to select among vehicles. For example, a user can request video for vehicles going between 25 and 40 MPH, or possessing the profile area of buses and trucks, to be output to the highlights video. The highlights video is meant to be used to sort through selected vehicles for the purpose of verifying speed tracking quality before making highlights public.

The final highlights video processor takes in a highlights video produced by VST and produces a final video file meant to depict carefully reviewed speeding vehicles. The highlights video processor supports user viewing of each vehicle as it passes through the speed measuring zone, replay, slow replay, deletion and keeping for the final highlights video. The final highlights video should be the sort of thing your lawyer would be comfortable with you posting to the internet.

Using VST

When you've got OpenCV dll's installed on your system, your camera with a street view set up, values for your particular scenario set in VST.cfg (e.g. frame counts for a calibration vehicle passing each way through the speed measuring zone), and video files your camera has captured, the next step for performing traffic speed analysis is to start up the videoSpeedTracker executable. Currently VST's routine steps on startup are carried out in a command line window (ripe opportunity for a GUI developer with time to do better). In sequence, you will first be shown the results of reading the VST.cfg file, then you will be asked to select a directory from the <path prefix> \\ IPCam directory, where you have done a one-time setup of the <path prefix> directory, e.g. "g:\locustData". Figure 1 depicts the steps I went through to select the directory "2016 02 05", located in "g:\\locustData\\IPCam" on my system, and containing all of my video files from February 5th, 2016. Next I selected the specific file "manual_20160205_115810.avi" for processing.

Next up is the question asking whether the Analysis Box (yellow region, as depicted in figure 2) and related components are OK. Currently, if you answer that the analysis box, etc. are not OK, the program aborts (because problems would be fixed by changing values in VST.cfg).



```
VideoSpeedTracker.exe
dataPathPrefix = g:\locustdata
L2RDirection = "SE"
R2LDirection = "NW"
obstruction[0] = 251
obstruction[1] = 311
AnalysisBoxLeft = 10
AnalysisBoxTop = 220
AnalysisBoxWidth = 1269
AnalysisBoxHeight = 190
speedLineLeft = 350
speedLineRight = 909
maxL2RDistOnEntry = 75
maxR2LDistOnEntry = 75
entryLookBack = 250
obstruction_extent = 30
largeVehicleArea = 100
CalibrationFramesL2R = 35
CalibrationFramesR2L = 40
SENSITIVITY_VALUE = 30
BLUR_SIZE = 20
SLOP = 15
R2LStreetY = 122
L2RStreetY = 158
nextHeight = 85

Want the directory 2016 02 04 <y/n> [n]:
Want the directory 2016 02 05 <y/n> [n]: y
Want the file manual_20160205_100732.avi <y/n/*> [n]:
Want the file manual_20160205_103644.avi <y/n/*> [n]:
Want the file manual_20160205_110553.avi <y/n/*> [n]:
Want the file manual_20160205_113338.avi <y/n/*> [n]:
Want the file manual_20160205_115810.avi <y/n/*> [n]: y

Are Analysis Box, Speed Measuring Zone,
Obstruction Framing, and Hubcap Lines OK <y/n> [y] ?
```

Figure 1. Command Line Setup



Figure 2. Yellow Analysis Box and Other Markings.

The yellow-edged analysis box depicted in Figure 2 serves a number of purposes: It crops out across the street neighbors' houses, it removes useless information in the foreground, and it greatly enhances the quality of the frame differencing method I use to detect motion: when the wind blows the leaves move, which creates bothersome noise in the difference image.

The white vertical lines are the end points of the speed measuring zone. A vehicle's speed is measured by counting how many $1/30^{\text{th}}$ of a second video frames it was in as it passed through the speed measuring zone. More specifically, the frame counter starts when the vehicle's front bumper first crosses the first white line in its path, and the frame counter stops when the vehicle's front bumper first crosses the second white line in its path. Its frame count is then compared to the frame count for a calibration vehicle that was driven through the same speed measuring zone at the speed limit (multiple vehicles, many times in my case, to be certain). A first order speed estimate for a vehicle analyzed by the Video Speed Tracker is derived by determining the value of:

Observed vehicle speed =

$$\text{int} (0.5 + ((\text{observed vehicle frame count}) / (\text{calibration vehicle frame count})) * \text{speed limit}).$$

Further analysis is performed in VST to home in on the likely actual speed but that analysis needn't be discussed here.

VST currently supports the description of one vertical obstruction in the foreground. In Figure 2 you can see the maple tree in my front yard, with two vertical yellow lines, one on either side of it. Because VST uses a frame differencing method to determine motion, it has to account for objects that will keep that method from working as needed. When the front bumper of a vehicle passes behind the tree, the frame differencing method will not detect the motion of the bumper. Because VST uses a “predictive tracker” this is not a problem: the occluded bumper is “dead reckoned” (and the rear bumper) as it passes behind the tree. If you have more than one foreground obstruction in your scenario, you’ll have to modify VST code a bit. It won’t be hard if you’re a programmer of any sort. I tell you what to do in the appendix. Currently, one obstruction can be characterized in VST.cfg.

Finally there are the hubcap lines: one horizontal orange line and one horizontal purple line as depicted in Figure 2. This is how I deal with shadows. In the bottom window (difference image) in Figure 3, you can see the “beaver tail” trailing behind the vehicle that is heading left-to-right. That’s its shadow moving. (In the same window you can see the tree passing up through the back part of the car’s hood –no motion detected behind the tree). There’s a lot of literature on detecting and removing shadows from video images. I read some and decided to hold off on any further consideration of shadow removal. The approaches are non-trivial. There’s an easier way for the straight, relatively flat stretch of road I’m analyzing: don’t look at the pavement in front of a car (It’s the shadows in front of a car that could cause problems for VST because they can interfere with quality front bumper tracking.) The orange (for right-to-left-traffic) and purple (for left-to-right traffic) lines are at roughly hubcap level. VST doesn’t consider motion information below the orange line for right-to-left vehicles, or the purple line for left-to-right vehicles. The bottom yellow line of the analysis box could be moved up to the purple line, but then the resulting video wouldn’t look as good.

If your street has a slope, bump or dip, the straight hubcap lines idea won’t work well. For a slope you could tilt your camera. For bumps or dips you’d need to describe a hubcap line with a polynomial. Then have VST sample the y value given the x coordinate of the tracked front bumper. That shouldn’t be hard. I just haven’t done it (another Github postable issue for VST). See the discussion of R2LStreetY and L2RStreetY in the appendix.

After accepting the Analysis Box, I was asked about a trace file, the speed limit, the egregious speed lower bound, the starting frame number to process, whether I wanted a highlights video file. For now, assume I said no to a highlights file.

Next, VST starts its analysis. Figure 3 depicts two new windows you’ll see open up. The top window is the region of the current video frame being analyzed by VST. As can be seen in the upper window in figure 3, each vehicle being tracked has two rectangles around it. A green rectangle depicts the predictive tracker’s expectation regarding the location of the vehicle. The blue line at the leading edge of the green rectangle is where the predictive tracker expects the front bumper to be. In order to do an

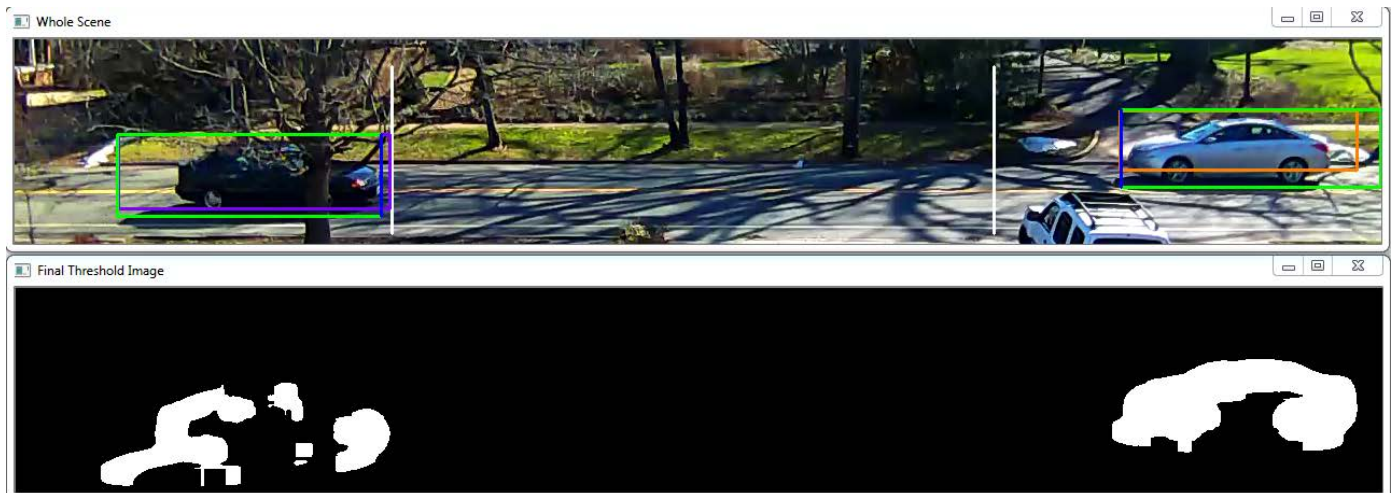


Figure 3. VST in Operation.

acceptable job of determining a vehicle's speed, the VST has to have locked the blue line onto the front bumper before the vehicle hits its first speed measuring zone line (the white vertical line). And it needs to be locked on when the vehicle crosses the second white line in its path. In actuality it only needs to be locked onto the same point on the vehicle as it crosses each white line, but VST attempts to make that be the front bumper. Also, the blue line doesn't have to be locked onto the front bumper between the white lines, although not being locked on at those times is rare if it's locked on at each of the white lines. Some drift may occur as two opposing vehicles pass by each other.

Vehicles traversing left-to-right also have purple rectangles around them, and right-to-left vehicles have orange rectangles. These rectangles represent what the current frame differencing data indicated about the current extent and position of the vehicle, as loosely constrained by the green box. You'd think the predictive tracker is failing if the blue line isn't coincident with the vertical purple (orange) line representing the leading edge of the vehicle. That's not always the case. The frame differencing operation can produce noisy data, for example when a front bumper passes behind a tree, when there's glare off a windshield, when the differencing operation fails some because there are pixels similar to the color of the vehicle, behind it, and others. This is why tracking radars like the ones used to track aircraft into and out of your local airport generally use predictive tracking also. The "real data" can be noisy, but you can generally pull enough useful data ("signal") out of the noise to get a very good representation of what's really going on. That's what VST's predictive tracker does. Usually, the blue line at the leading edge of the green prediction rectangle is telling the truth better than the current snapshot of "real data" gotten from the frame differencing.

VST really only needs to track the front bumper of a vehicle to get speed results. Tracking height and rear bumper are done only to support computation of profile area (to separate buses from cars from motorcycles) and to handle issues that arise when two opposing vehicles pass each other.



Figure 4. Opposing cars in the speed measuring zone.

There are two decent technical accomplishments in VST: the quality of its predictive tracker (look where the blue lines are in Figure 4) and its ability to continue to track opposite direction passing vehicles with high probability of success. The difference images for the two vehicles in Figure 4 are about to collide. Without predictive tracking, all would most likely be lost. Figure 5 shows what happens next.

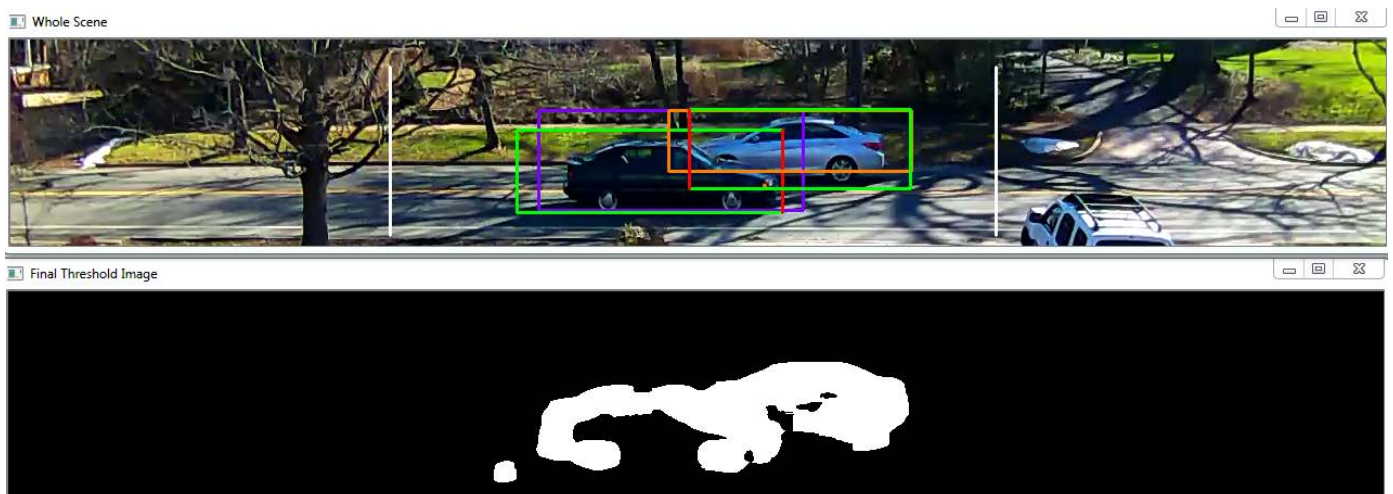


Figure 5. Dead Reckoning Front Bumpers of Opposing Direction Passing Vehicles.

As soon as a pair of front bumpers of opposing vehicles passes each other, VST turns the leading blue lines red to indicate that each vehicle is being dead-reckoned, meaning that predicted data is being used as input to the predictive tracker rather than actual data derived from the difference image. Without performing some serious image analysis on the blob depicted in the bottom window of Figure 5, it's essentially impossible to pick out the front bumpers of the two vehicles. The quality of the prediction is better than the quality of any easily analyzed real data at this point.

As soon as the front bumper of a vehicle proceeds past the predicted rear bumper of the opposing vehicle, VST starts using the difference image data again to feed the tracking filter for the front bumper of the first vehicle. You can now see why locations of rear bumpers need to be known: so that VST can determine when to turn off dead reckoning a front bumper.

VST also, separately, dead reckons rear bumpers as needed. Figure 6 shows the dead reckoned rear bumpers of two passing vehicles at the last moments of their rear bumpers being dead reckoned. VST stops dead reckoning rear bumpers when it determines the predicted positions of the rear bumpers are past each other. Notice how tightly the tracker lock has stayed on the front bumpers despite the vehicles' recent passing.

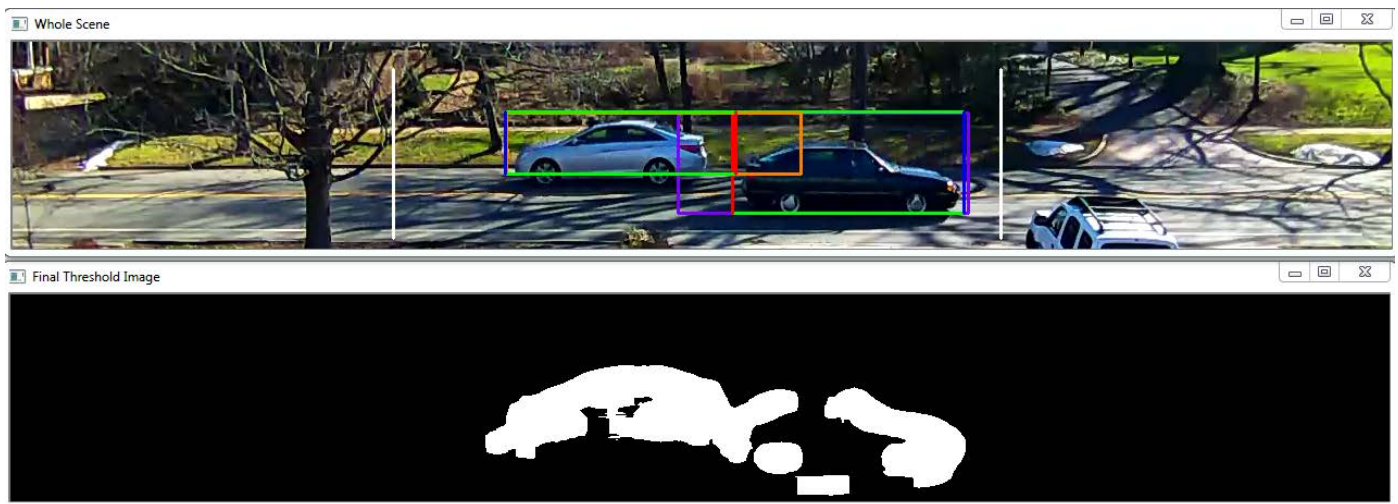


Figure 6. Dead Reckoning Rear Bumpers.

Finally, as seen in Figure 7, the vehicles have crossed their respective second speed measuring zone white lines, VST has computed their respective speeds, automated quality checks have been passed, and speeds are posted. And so it goes.

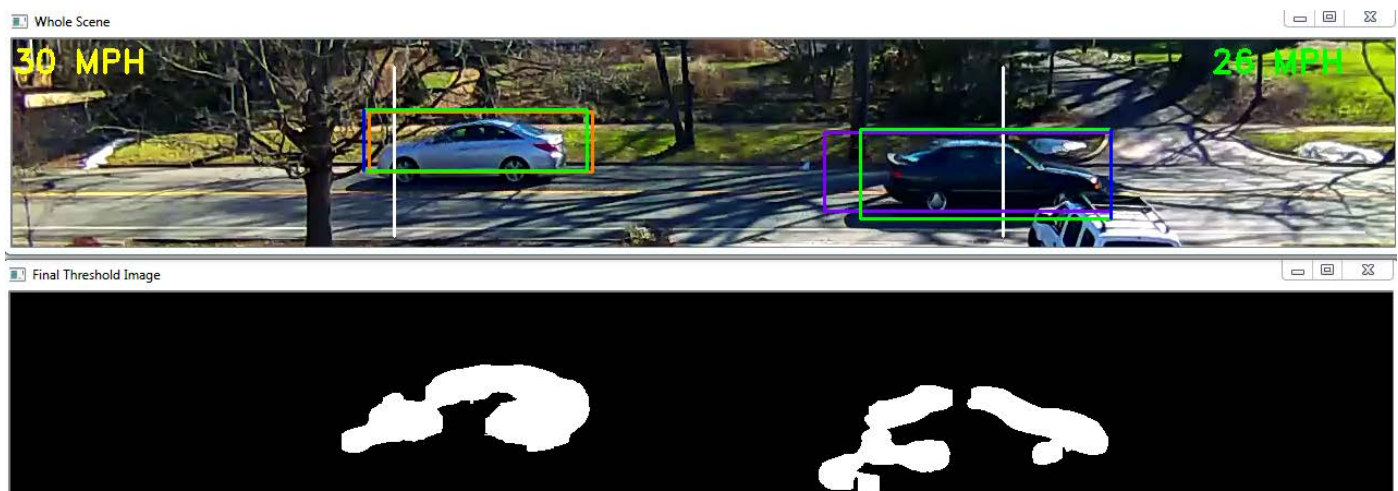


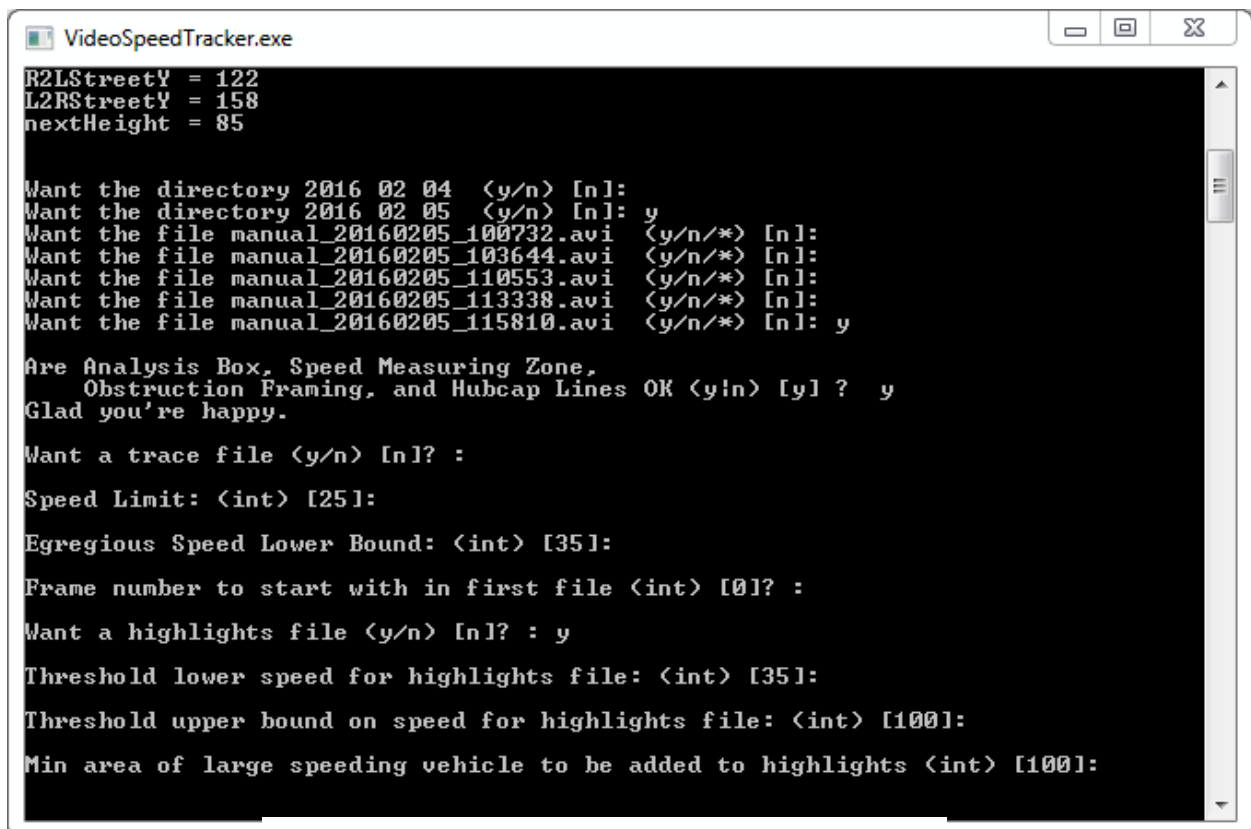
Figure 7. Speeds Determined for Both Vehicles.

While the VST is executing, the user has various options. When the analysis window (e.g. the top window in Fig. 7) is highlighted, the user can enter a small set of single letter commands to change processing behavior:

1. “p” will toggle pausing the analysis and resuming it
2. “v” toggles display of the lower window in Fig. 7, and stops refresh of the upper window, allowing processing to proceed faster (VST tends to be I/O bound).
3. “f” makes the display update of the two windows in Fig. 7 occur five times faster down to a delay lower limit of 10. Consequently the execution speed of VST goes up accordingly.
4. “s” makes the display update of the two windows in Fig. 7 occur five times slower up to a delay upper limit of 1250. Consequently the execution speed of VST goes down accordingly.

Producing a Highlights Video File in VST

You’re given an option to have a highlights video file produced as a byproduct of the execution of VST. In Fig. 1 the user said “no” to such a highlights video file. Figure 8 depicts the interactive session when the user says “yes”. Consequently the user is given the chance to enter qualifying information for vehicle speeds, and profile area, leading to the vehicle’s inclusion in the highlights video file.



```
VideoSpeedTracker.exe
R2LStreetY = 122
L2RStreetY = 158
nextHeight = 85

Want the directory 2016 02 04 <y/n> [n]:
Want the directory 2016 02 05 <y/n> [n]: y
Want the file manual_20160205_100732.avi <y/n/*> [n]:
Want the file manual_20160205_103644.avi <y/n/*> [n]:
Want the file manual_20160205_110553.avi <y/n/*> [n]:
Want the file manual_20160205_113338.avi <y/n/*> [n]:
Want the file manual_20160205_115810.avi <y/n/*> [n]: y

Are Analysis Box, Speed Measuring Zone,
Obstruction Framing, and Hubcap Lines OK <y/n> [y] ? y
Glad you're happy.

Want a trace file <y/n> [n]? :
Speed Limit: <int> [25]:
Egregious Speed Lower Bound: <int> [35]:
Frame number to start with in first file <int> [0]? :
Want a highlights file <y/n> [n]? : y
Threshold lower speed for highlights file: <int> [35]:
Threshold upper bound on speed for highlights file: <int> [100]:
Min area of large speeding vehicle to be added to highlights <int> [100]:
```

Figure 8. Requesting Highlights Video File in VST.

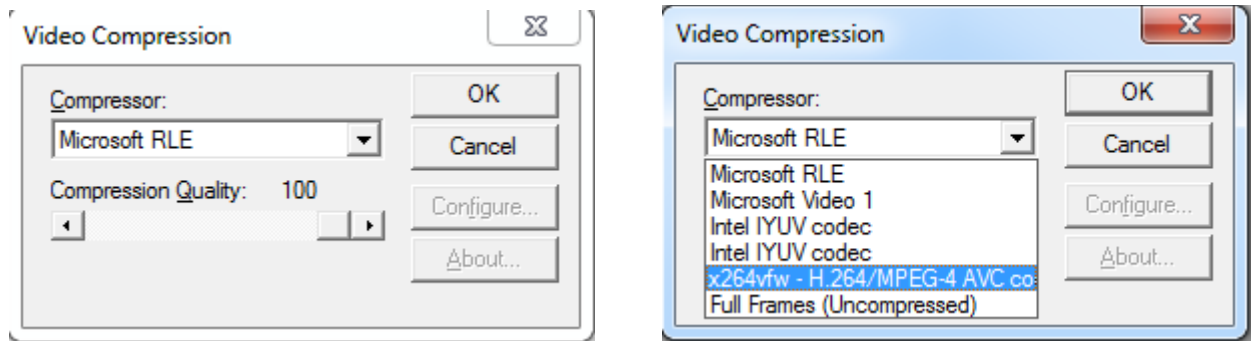


Figure 9. CODEC Selection.

After answering questions about the highlights video selection data, a video compression selection box will pop up, as depicted on the left in Figure 9. You need to have installed the h264 codec or whatever codec you choose to use. It is strongly advised that you use a codec that does high compression or you will produce a very large highlights file (orders of magnitude larger than what h264 produces). Scroll down in the list and select the codec you choose to use and select OK. The VST will start up, and you will see the two new windows depicted in Figure 3. Sorry about the pop-up box. OpenCV 2.4.11 has a small bug that prevents specification of an h264 codec in the open file operation.

Further Notes on the VST

Every once in a while a horizontal red line will pop up through the middle of the Analysis Box. This means that VST has determined there's too much going on to produce valid results with high likelihood. One threshold for abandoning current analysis is that there are two opposite direction vehicles detected, as well as a third vehicle, and a fourth has just appeared. Four vehicles, with at least two opposing, means that too much dead reckoning will have to be done. This can be visually confirmed by inspecting the scene in any of figures 3 through 7. VST can handle an arbitrary number of vehicles all traveling in the same direction.

VST also abandons (red lines) all tracking when it detects that one vehicle is overtaking another moving vehicle (both heading in the same direction). This happens on my street when a vehicle passes a bicycle. (Yes, VST tracks bicycles.) It would be preferable to keep both vehicles in track, but that's harder than it may first seem. They could be leapfrogging each other. (Another Github issue for consideration.)

Sometimes VST will fail to track a vehicle well. There are many circumstances that can cause this outcome: a vehicle too close in color to the background behind it, low quality differencing images for any of many other reasons, too much dead reckoning past a very large vehicle too soon after the vehicle was first put in track, etc. VST attempts to identify bad tracks but doesn't always catch them. That's why there's a post processor for the highlights video.

VST abandons track of a given vehicle when it detects that vehicle is moving backwards relative to what VST was expecting. VST abandons track when, twice in a row, it doesn't see any frame differencing image where a vehicle is expected to be.

VST is aggressive about keeping moving objects in track, even if it lost track on them previously. This is done primarily to keep the quality of opposite direction tracking high. Bad things happen if a tracked vehicle passes another when the tracker doesn't know the second one is there. You'll see evidence of VST's aggressiveness on occasion, when track is lost on a vehicle, and then picked right back up the next frame. If the vehicle is more than half way across the analysis box VST will try to place it in track in the opposite direction it's traveling. Soon after, track will be dropped, for backwards motion, and then picked right back up again. No need to worry. All of those futile attempts are thrown away.

If a track is lost for a vehicle when its front bumper is in the speed measuring zone, no speed for that vehicle will be entered into final statistics. No vehicle that is first put into track after its front bumper has passed the first white speed measuring marker will have its speed entered into final statistics. In general, if a speed is posted for a vehicle in the upper corner of its destination end of the analysis box, then that speed is recorded into the spreadsheet (csv) data file. Posted speeds can be seen in the top window of Fig. 7.

Post-Processing with the Final Highlights Video Processor

Input to the second program, the final highlights video processor (FHVD), is a video highlights file produced by VST. The purpose of the FHVD is to provide the user with an opportunity to hand select examples of speeding vehicles for public analysis and review, and an opportunity to ensure no speeding mistracked vehicle is ever presented as having been tracked correctly.

When the FHVD is started it lists the avi files found in the HiLites subdirectory, one-by-one, as can be seen in Figure 10. You select the file to be edited, and then confirm the cropping box is acceptable, see figure 11. Note the cropping box is 640 x 360 pixels. This smaller size is used to make posts of the output file to the web more easily viewed. 1280 wide files do not show well on Facebook, for example.

Next, the codec selection popup will occur, as seen in figure 9. The codec you choose will be used to encode the edited highlights video that FHVD outputs. Once you've chosen the codec (h264 highly recommended), the editing process will begin.

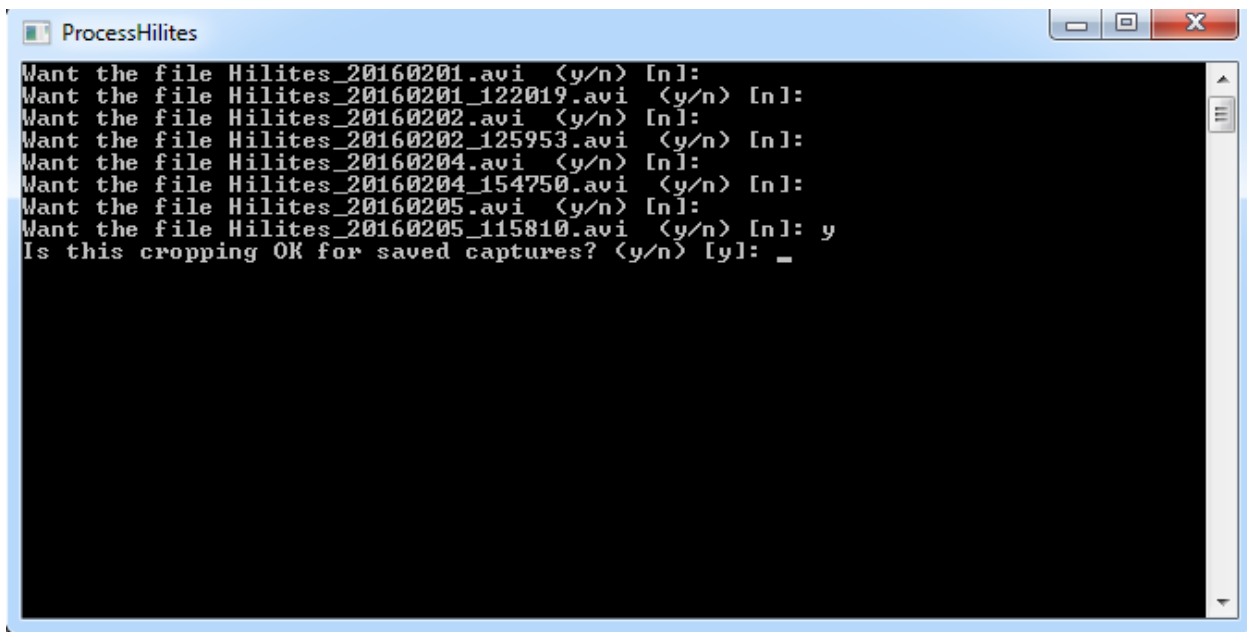


Figure 10. FHVD Input File Selection.

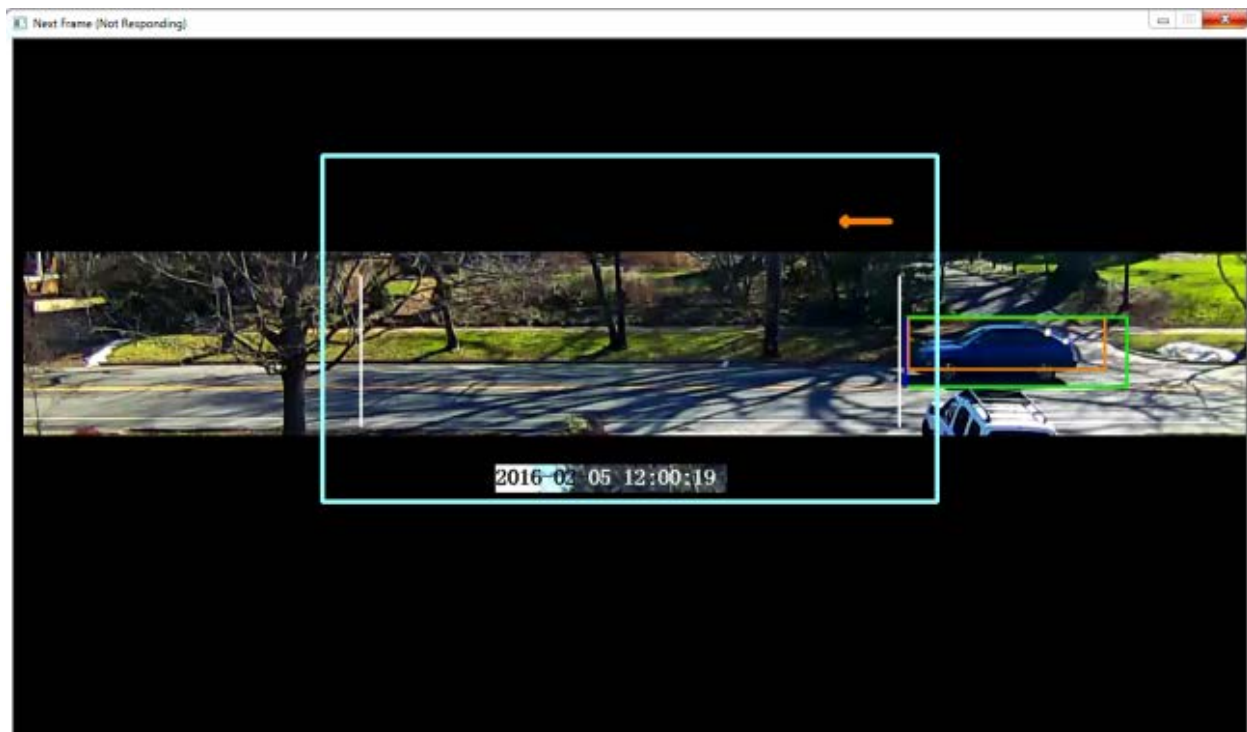


Figure 11. Cropping Box for FHVD Output

FHDV presents the motion of vehicles, one at a time, with the last frame displayed being the one showing the vehicle across the end of the speed measuring zone, and with the speed of that vehicle shown above it. At this point you have the following options:

1. [R]eplay – Show the motion of the vehicle through the speed measuring zone again.
2. [S]low replay – Do a slow motion replay.
3. [D]elete – Do not write the current vehicle motion to the output file. Proceed to next vehicle.
4. [K]eep – Do write the current vehicle motion to the output file. Proceed to the next vehicle.
5. [Q]uit – Stop processing vehicles.

FHVD's output file has the prefix "forPost" and a suffix taken from the original video file processed by VST. The forPost file is written to the subdirectory <path prefix>\HiLites\forPosting.

Current Development Environment and Implementation

Both VST and FHVD have been developed in M.S. Visual Studio 2013, C++, using the OpenCV C++ API on a Win7, 64 bit platform. I have made every effort to minimize system dependencies. If you're in a different OS, you'll need to address backslashes in path names and two system calls in setup() in trafficReports.cpp, where I make sys() calls to get the file names in a directory. You'll probably find some other dependencies as well.

You also need to have OpenCV 2.4.11 installed. You can find the download you need here: opencv.org/downloads.html. I first tried using the then relatively new 3.x version and ran into too many bugs. Your mileage may vary. Once I got the installation right, OpenCV 2.4.11 has worked without noticeable bugs (except the codec pop-up issue noted above).

Installing Executables

Executables that can run on Win7tel64 platforms can be found in the repo bin directory. You'll find two executables and two config files that should co-reside with them, wherever you put them. To get the executables running:

- Install OpenCV 2.4.11. opencv.org/downloads.html
- Edit your path variable to include the path to OpenCV's x64 bin directory full of OpenCV dll's. My directory is at G:\builds\opencv2411\build\x64\vc12\bin
- Set up the four VST directories (and one subdirectory) as spelled out in part 6 of the Set-up checklist following.
- Edit the two config files (VST.cfg and ProcessHiLites.cfg), changing the value for the first entry in each, "dataPathPrefix", to the directory in which you placed the five directories you created in the previous step.
- Feel free to use the avi file I've provided in the repo sampleData/IPCam directory.
- Execute!

Set-up Checklist

- 1) You really want 100 feet or more of open view (a maple tree trunk in the foreground is OK) of the street. Your speed measuring zone should be at least one second wide for a vehicle going the speed limit. For a speed limit of 25 MPH, that's about 37 feet. My speed measuring zone is 1-1/3 seconds wide, which translates to about 50 feet for a vehicle going 25 MPH. I've been able to track vehicles up to 71 MPH (Blue line on the front bumper the whole way!) in the zone you see in Figure 2.
- 2) You need an HD (1280 x 720) video input stream for the stretch of street you want to analyze. That requires a camera. I'm using the Foscam FI9803EP outdoor, HD, power over ethernet camera. It's about \$90 online. I have no particular loyalty to Foscam, but I can say I've had about a half dozen of their cameras and only one (an FI9821) has developed an issue (when it pans, the video signal cuts out). That's a decent record for an inexpensive cam that does HD and 30FPS. My 9803EP has been working about three weeks, including through the infamous Middle Atlantic January 2016 Snowzilla.
- 3) Aim the camera well. Look at Figure 2. Aim the camera so that the street passes left to right about half way up the overall image. This should minimize the effects of lens distortion. A vehicle moving a constant velocity cross the lens moves at different pixel rates, frame to frame, as it proceeds from an edge towards the center and back towards the far edge. My tracker is what's known as a "piecewise, linear least squares" tracker. The piecewise part (throwing out the oldest data) is what allows it to adapt to changing pixel rates as a vehicle moves through the scene. Setting up as you see in Figure 2 helps the tracker produce high quality results.
- 4) If you're recompiling, you'll need OpenCV 2.4.11 installed to have a successful compile. If you're executing a provided executable you'll still need OpenCV 2.4.11 runtime libraries (dll's in Windows) before you can execute successfully.
- 5) Set critical one time setup values in VST.cfg. These values are described in the appendix.
- 6) Create a location (the "prefixPath") for five directories used by VST (and the final highlights video processor):
 - a. IPCam -- You put directories for (typically) daily collections of .avi video files here.
 - b. HiLites -- VST outputs highlights videos into this directory
 - i. forPosting -- Subdirectory for output of final highlights video processor
 - c. Stats -- VST puts csv files with tracked vehicle data here
 - d. Trace -- VST puts debug files here.

All of the files VST creates are given names derived from the input video file name (or the video file directory if you answer "*" when asked for which file in a directory full of video files to process.) For the IPCam directory, I create subdirectories with syntax yyyyymmdd (e.g. 20160205) in which I place that day's video files.

Appendix

Objects used in VST that need to be set in VST.cfg

VST.cfg has been created to maximize the chances of being able to distribute an executable (i.e. not requiring a recompile). VST.cfg syntax is a little fragile. Currently you can't have any blank lines, comment only lines, or extra lines at the end (Github issue to make VST.cfg more forgiving). Each line has the syntax:

<known object name> = <value> #comment

Actual comments at the ends of lines can be omitted, but not the "#". My parser looks for it. Also, the known object names must appear in the order they appear in the distributed VST.cfg file, and as they appear in the following table. Spaces are allowed on either side of the "=".

Known object	Purpose
dataPathPrefix	Needs to be set to the root directory containing data directories for VST. The required directories in this root are: "IPCam", "Stats", "Trace" and "HiLites". HiLites should have a subdirectory "forPosting". Note: on Windows, do not place any double back slashes in the path name. They're not needed here.
L2RDirection	String capturing compass direction of L2R vehicles.
R2LDirection	String capturing compass direction of R2L vehicles.
obstruction[]	Two element vector holding x coords of left and right side of one obstruction. Vehicle motion code will dead reckon front bumper between these x values inside analysis box. If you have no obstructions (e.g. a maple tree between camera and street) set second value smaller than first. If you have more than one obstruction, you'll need to make obstruction[] two dimensional and change checks in vehicle dynamics to loop over multiple obstructions. Syntax of RHS value pair is "[<left> , <right>]"
AnalysisBoxLeft	x coord of speed analysis box. Note the origin in OpenCV is top left. X values go right, Y values go down.
AnalysisBoxTop	y coord of top of speed analysis box, with top left as origin. Note the origin in OpenCV is top left. X values go right, Y values go down.
AnalysisBoxWidth	horizontal width of speed analysis box. Note the origin in OpenCV is top left. X values go right, Y values go down.
AnalysisBoxHeight	vertical height of speed analysis box as measured down from AnalysisBoxTop. Note the origin in OpenCV is top left. X values go right, Y values go down.
speedLineLeft	x coord of left end of speed measuring zone (left vertical white line).
speedLineRight	x coord of right end of speed measuring zone (right vertical white line).
maxL2RDistOnEntry	Used to determine how aggressively tracker should look out in front of *entering* vehicle to ultimately be able to lock onto front bumper for L2R vehicle. Units are pixels.
maxR2LDistOnEntry	Ditto for R2L vehicle. Units are pixels.

entryLookBack	Used to determine how far back to look for rear bumper of entering vehicle. Units are pixels. Making this value too large can create large tracked area (rearward extent of green box) behind entering vehicle that can cause excessive dead-reckoning of passing, opposite direction vehicles. Large vehicles may not have their rear bumpers found when entering. Often rear bumper tracker will slowly find actual rear bumper of large vehicles after front bumper has entered beyond entryLookBack pixels. Still fails occasionally on UPS trucks and long single color panel trucks. An improved solution, though low priority, is needed.
obstruction_extent	Even after a vehicle has passed an obstruction it needs additional span to show up as a blob past the obstruction in frame differencing operation.
largeVehicleArea	When choosing vehicles for the highlights video, this value is consulted to determine if the vehicle's (normalized) area exceeds it. All computed areas are normalized so that buses and large trucks tend to be most of the vehicles over 100, which is this variable's default value. Your value may vary.
CalibrationFramesL2R	Turn your camera on in record mode and drive a vehicle left to right through the scene. Try to go 25MPH. Come back home and set this value to $(\text{speed limit} / \text{actual speed}) * \text{number of frames it took for you to drive through the zone in which you'll be measuring speed.}$. It would be better to drive through multiple times with multiple vehicles to get best calibration.
CalibrationFramesR2L	Same as process for L2R, but R2L. Note that due to parallax, the R2L (farther away) lane will require more frames to travel through the speed measuring zone at a given speed (e.g. <speed limit>) than will be seen for L2R calibration. My values are L2R: 40 frames, and R2L: 36 frames for about 50 feet of speed measuring zone and 25 MPH speed limit.
SENSITIVITY_VALUE	Sensitivity value for the OpenCV absdiff() function. It probably doesn't need to be tweaked, but it is a fine tuning parameter for the speed tracker. I set it early on and forgot it.
BLUR_SIZE	Used to smooth the intensity image output from absdiff() function. Again, I set it once and didn't need to tweak it.
SLOP	Margin of error allowed (units are pixels) in predictive tracker when trying to determine if two opposite direction vehicles are overlapping.
R2LStreetY	y coordinate in the analysis box for describing R2L vehicle hubcap line. Currently a constant because I have a flat, non-sloping street. Slopes, bumps and/or dips could be described by changing R2LStreetY to a function of x, where R2LStreetY() describes an arbitrary polynomial you provide.
L2RStreetY	y coordinate in the analysis box for describing L2R vehicle hubcap line. Currently a constant because I have a flat, non-sloping street. Slopes, bumps and/or dips could be described by changing L2RStreetY to a function of x, where L2RStreetY() describes an arbitrary polynomial you provide.
nextHeight	The value of this variable assumed before an actual vehicle height estimation can be conducted is a constant in the code. You probably won't have to change it in your setup, but you might. Once three or more differencing operation images are produced for a vehicle entering the scene, height will be calculated from data.