LLNL-CONF-413376

Ultra-scale vehicle tracking in low spatial-resolution and low frame-rate overhead video

C. J. Carrano

May 28, 2009

Signal and Data Processing of Small Targets 2009 in the SPIE Symposium on SPIE Optical Engineering + Applications
San Diego, CA, United States
August 2, 2009 through August 6, 2009

Ultra-scale vehicle tracking in low spatial-resolution and low frame-rate overhead video

Carmen J. Carrano*[a]
[a]Lawrence Livermore National Laboratory, 7000 East Ave., Livermore, CA, USA 94550

ABSTRACT

Overhead persistent surveillance systems are becoming more capable at acquiring wide-field image sequences for long time-spans. The need to exploit this data is becoming ever greater. The ability to track a single vehicle of interest or to track all the observable vehicles, which may number in the thousands, over large, cluttered regions while they persist in the imagery either in real-time or quickly on-demand is very desirable. With this ability we can begin to answer a number of interesting questions such as, what are normal traffic patterns in a particular region or where did that truck come from? There are many challenges associated with processing this type of data, some of which we will address in the paper. Wide-field image sequences are very large with many thousands of pixels on a side and are characterized by lower resolutions (e.g. worse than 0.5 meters/pixel) and lower frame rates (e.g. a few Hz or less). The objects in the scenery can vary in size, density, and contrast with respect to the background. At the same time the background scenery provides a number of clutter sources both man-made and natural. We describe our current implementation of an ultra-scale capable multiple-vehicle tracking algorithm for overhead persistent surveillance imagery as well as discuss the tracking and timing performance of the currently implemented algorithm which is aimed at utilizing grayscale electro-optical image sequences alone for the track segment generation.

Keywords: vehicle tracking, persistent surveillance, low spatial-resolution, mover segmentation, low frame-rate, airborne imaging

1. INTRODUCTION

Tracking multiple vehicles in persistent surveillance image sequences is a very difficult problem in general for a number of reasons relating either to the scenery or the sensor.

Scenery issues:

- Small vehicles (e.g. several to tens of pixels) with sometimes low (and changing) contrast
- Many vehicles, some of which may be close together, with variable speeds and paths including stops.
- Obscurations of the path by buildings, overpasses, trees, terrain, or other vehicles.
- Other scene clutter that may appear to be moving such as sun glints off water or other reflective surfaces (e.g. parked cars), tall building edges, etc.

Sensor issues:

- Low frame-rate and/or low spatial resolution
- Limited spectral information (e.g. grayscale only, …)
- Image stability issues, errors in navigation data
- Camera noise both static and dynamic

Our current tracking approach relies on the mover map, path dynamics, and image features to perform tracking. The benefit of using image features in addition to the mover map is that we can not only track vehicles when they move, but also when they stop. The algorithm does very well when the vehicles are sufficiently separated and the obscurations are small enough such that the vehicles keep a constant speed and direction under the obscuration.

*carrano2@llnl.gov

As with any tracker operating on difficult data, there are a number of cases where tracks have a higher probability of ending prematurely (complicated dynamics situations, big obscurations, vehicle-like clutter, low-contrast vehicles, high-traffic density, etc.) One option is to consider this a first stage tracker that generates track segments that can be linked together in a later stage of tracking either automatically or with the human-in-the-loop for high-value targets.

Another facet to our approach is that we wanted to keep it relatively simple enough to allow for parallelization and real-time operation up to certain image sizes and vehicle counts, where real-time means keeping up with the camera frame-rate (e.g. 2 Hz). Our near term goal requires real-time operation on image sizes up to 2000x2000 pixels tracking several hundred vehicles at a time with an eventual goal of operating on image sizes up to 4000x4000 pixels or more simultaneously tracking on the order of one thousand vehicles with a 0.5 to 1.0 meters-per-pixel ground-sample interval.

Section 2 describes the necessary pre-processing prior to tracking, Section 3 describes the tracking, and Section 4 covers the performance and example results.

2. PRE-PROCESSING

The basic processing steps required on raw camera data before tracking is even possible are given in Figure 1. This assumes the raw data is calibrated and free from strong artifacts. If that is not the case, then extra image conditioning steps are preferable for best results.

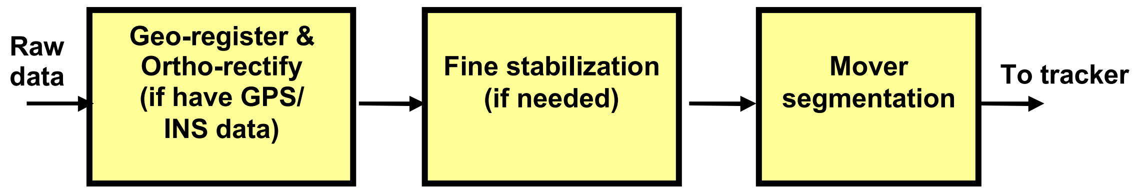

Raw data
Geo-register & Ortho-rectify (if have GPS/ INS data)
Fine stabilization (if needed)
Mover segmentation
To tracker

Figure 1: Basic processing steps prior to tracking

2.1 Geo-registration and ortho-rectification

If streamed GPS and inertial navigation system data are available with the image sequence, it is possible to geo-register and resample the data to a NADIR looking view at a specific and constant ground sampling interval. Because it is a simple geometric transformation, this step can be done very quickly using graphics-processing-units (GPUs). Pixel positions and hence tracks from this data are straightforward to translate to latitude and longitude for later integration into geographical information systems or spatial databases holding the track segments. If more than one camera is being used to create an even wider-field image, then image stitching can also be performed just prior to this step. The algorithm and GPU implementation for doing these operations are beyond the scope of this report but can be found in [2].

2.2 Stabilization

As with most real-life navigations systems, the navigation data used in performing the geo-registration is subject to errors. Likewise, the cameras won't necessarily be perfectly calibrated in position. As a result, the geo-registered and ortho-rectified image sequence will not necessarily be suitably stabilized for optimal mover segmentation. If we can stabilize the image down to single pixel accuracy we will have optimal conditions for the mover segmentation step. Typically, for airborne imagery an affine-based stabilization works very well but a dense correspondence approach can also be used, especially if there is significant terrain relief or tall buildings. The stabilization algorithm we commonly use is described in [3] but can be found in common textbooks on the subject [4]. Work is also being done on GPU based stabilization which we intend to utilize in our processing pipeline to improve speed [5].

2.3 Mover segmentation

The mover segmentation is a very important step because the detected motion regions are used directly to guide the tracker. The purpose here is to generate a binary image for each intensity image in the sequence where a "1" indicates a motion region and a "0" does not. With the slower frame rates (~couple of Hz) of our available data, our current preferred approach is a difference-from-median based approach. It is a non-recursive, highly adaptive technique that uses a sliding time window. The user can pick the frame-buffer (i.e. time window) size. With the lower frame rates, buffer sizes of 5 to 7 frames work well. Although increasing the buffer size helps with slower movers, we have to be careful for two primary reasons. First, with larger images we might need to be careful about the memory usage and algorithm speed as the frame buffer increases. The other reason is that all the details of the imagery won't necessarily be stabilized for long time periods due to parallax for taller structures and slow drifting, both of which can give false mover detections.

The algorithm is conceptually quite simple. A block diagram is shown in Figure 2. An estimate of the stationary background at frame N is obtained by computing the temporal median at each pixel location over some number of frames, B. The absolute value of the subtraction of the background estimate from the current (foreground) frame N yields an absolute difference image where brighter regions have a greater probability of being in motion and darker regions are less likely to be in motion. At this point, a threshold based on the statistics of the difference image is applied. Typically 5*sigma to 8*sigma is good for selecting the motion regions. We have built in the ability to let the threshold vary spatially, so instead of calculating and using the sigma of the whole image at once to determine a single threshold, we can calculate it locally with some given region size. This would be useful for images whose mean intensity has some large scale variations. After the threshold step, we have always found it useful to reject all regions less than or greater than a certain pixel count. The pixel counts depend on the resolution of the imagery. Sometimes we also find it useful to apply a morphological closing operation with a small square or circular mask to clean up the blobs. Example imagery at each major step of the mover detection procedure is shown in Figure 3.

Depending on how well the stabilization was done and the scenery content, what we see in the mover map may be exactly what we wanted (moving vehicles) or it may contain a number of clutter sources. These clutter sources come from buildings, especially tall building parallax, terrain altitude variations, other moving objects (e.g. trees in the wind), noisy camera data, and sun glints off water or parked cars. There are various ways to deal with the clutter sources. Ideally, any artifacts related to the camera should be cleaned up prior to any processing. Sometimes clutter can be cleaned up with the region size filter, but often these clutter sources will make it through the region size filters. A possible way to deal with movers not associated with vehicles on roads is to mask out regions where vehicles can't go. The challenge would then be generating the road network mask from the data itself or ancillary information which may or may not be available.

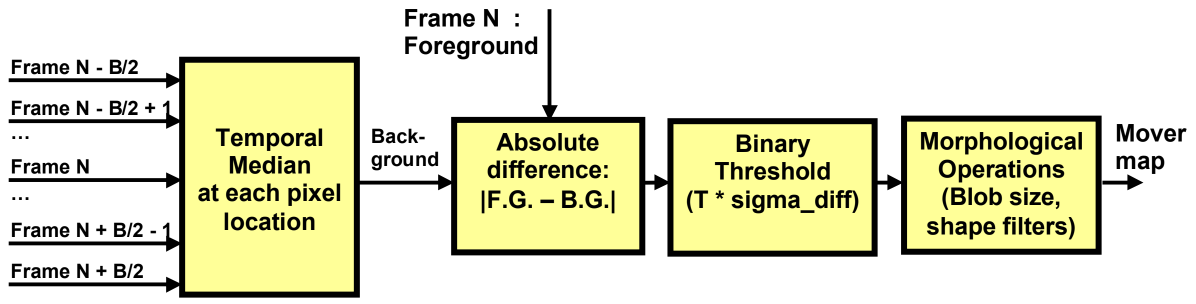
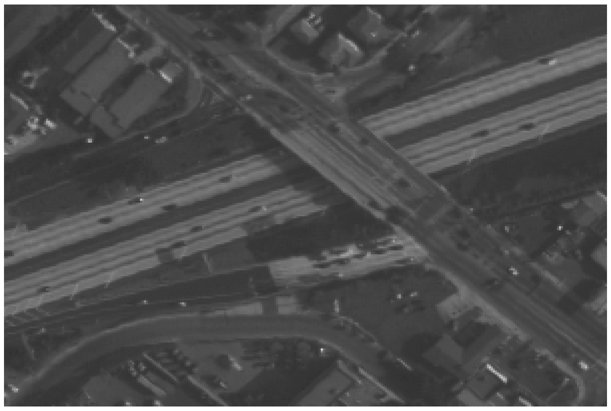

Frame N - B/2
Frame N - B/2 + 1
...
Frame N
...
Frame N + B/2 - 1
Frame N + B/2
Temporal Median at each pixel location
Back-ground
Frame N : Foreground
Absolute difference: |F.G. – B.G.|
Binary Threshold (T * sigma_diff)
Morphological Operations (Blob size, shape filters)
Mover map

Figure 2: Block diagram of median-filter based motion segmentation.



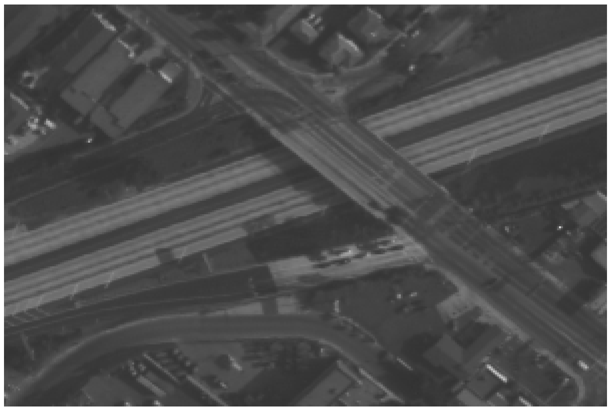


Figure 3a: Foreground frame (a small region)

Figure 3b: 5-frame temporal median or background frame

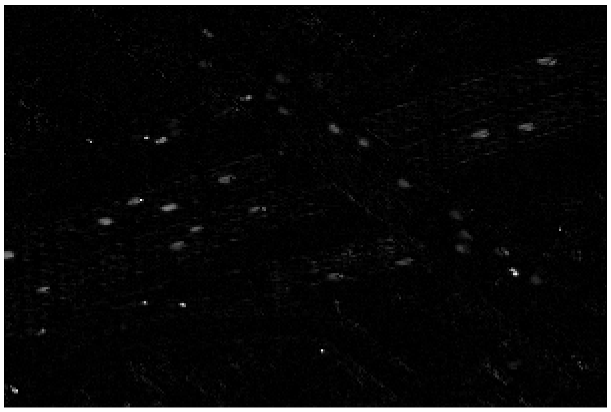


Figure 3c: Absolute difference

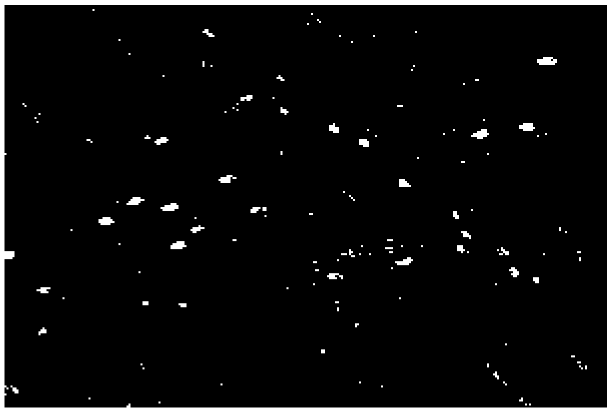


Figure 3d: Thresholded

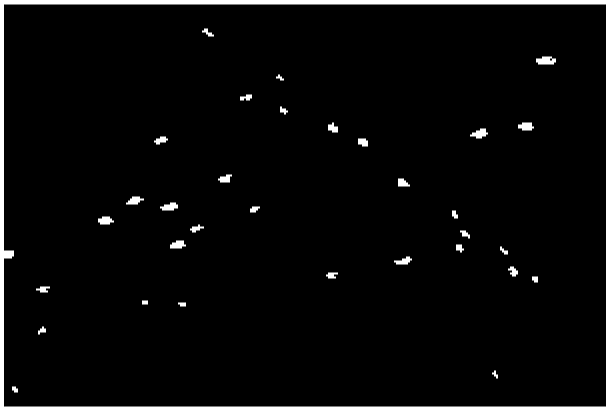


Figure 3f: Region-size filtered

Figure 3: Example of mover detection processing.

3. ULTRA-SCALE TRACKING ALGORITHM

3.1 Track initialization

Before vehicles can be tracked, they must be found in the first place. Ideally, we would like to have a perfect vehicle detector to initialize all the potential tracks, though many of the detected vehicles would be stationary and of little interest for tracking. Our track initialization approach is simple; on the first frame of interest, all validly detected motion regions are considered to be candidate movers and are initialized with a time, frame number, position, bounding box, zero velocity and a "new track" status. The left side of Figure 4 shows seven detected moving regions from a small section of the first frame of an image sequence.

Because parked vehicles can start moving at any time and vehicles can move in from the edge of the imagery, with each new frame it is necessary to check if there are new movers to initialize. As the track association proceeds, all movers that have been accounted for are zeroed out in the mover map and any remaining movers are then initialized as new movers. The right size of Figure 4 shows the same seven regions along with two newly detected movers five frames later.

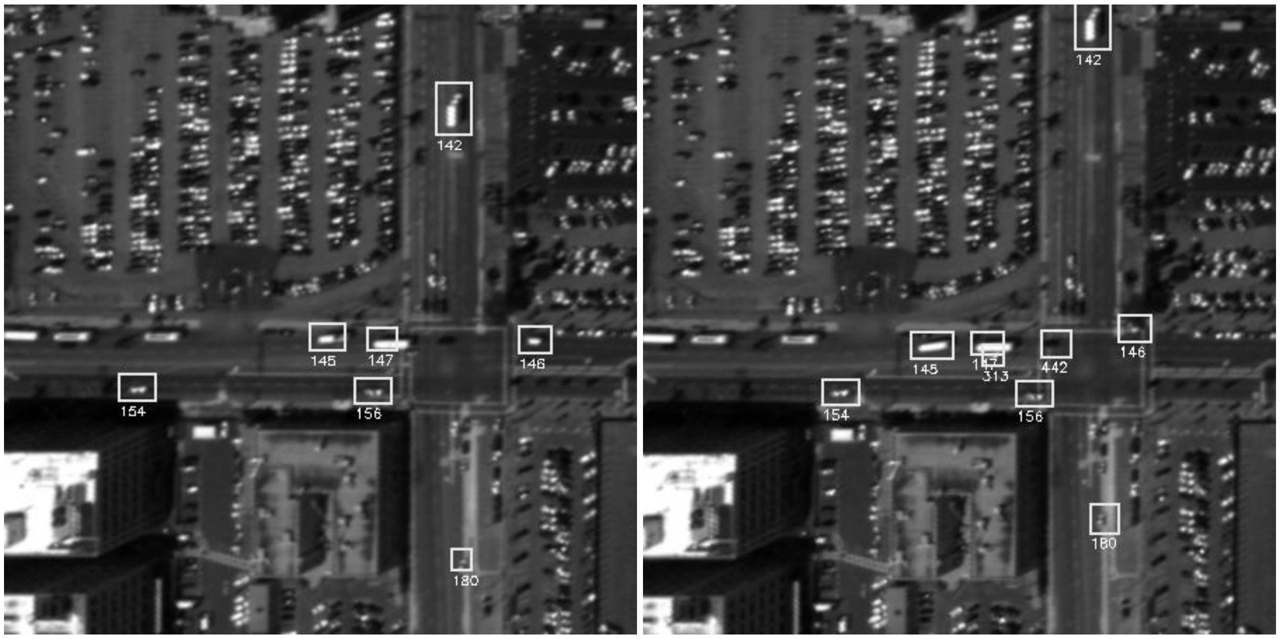


Figure 4: (Left) Newly initialized tracks shown with initial bounding boxes and track ID's from a small section of imagery. (Right) Same region of data 5 frames later with two new movers detected (313 and 442).

3.2 Track association for new tracks

Once new tracks are initialized, we only have a single frame of observation, which means we don't know the speed or direction of the vehicle yet. To accommodate this, we allow for a larger search space than usual based on a maximum allowed speed and we only use target features for association. The two features we currently use are the peak of the phase cross-correlation [7] and an average intensity difference between the current vehicle and the candidate vehicle in the next frame. The candidate target with the highest phase cross-correlation peak whose intensity difference is low enough gets the match. The matching vehicle then gets a track status of moving, a track length of 1, a new position, a new bounding box, and a velocity assigned to it.

We allow the bounding box to slowly change size as the detected region size or shape changes. This is useful when the vehicle speeds up to reveal its true size or when a longer vehicle turns a corner. There is sometimes confusion when a vehicle slows down because the motion region will either shrink (on small vehicles) or break up (larger vehicles). One solution not yet implemented would be to keep the vehicle area constant once we are sure of its footprint area and not let it shrink.

3.3 Track association for existing tracks

Now that we have the initial velocity estimate for each track, we can predict the next position using our constant-velocity dynamics model,

$$X_{pred}[t]= X[t-1] + V[t-1] * \Delta t, \qquad [1]$$

where X is the position vector, V is the velocity vector and Δt is the time between frames. The tracker loops through each vehicle in motion and tries to find the next location of the vehicle. Using the mover map, it only searches motion regions that fall inside its allowable search area. The parameters to calculate the size of the search area are defined at the start of the algorithm based on realistic maximum accelerations and decelerations we would expect from a vehicle as well as allowable angular deviations from a straight line in a given time step. We have used varying search area shapes from circles or annular cones in the IDL codes to squares in the real-time code depending on how much computation we want to perform. It is simple to modify the tracker to vary the search area methods or use just one of them. A diagram of the methods is shown in Figure 5.

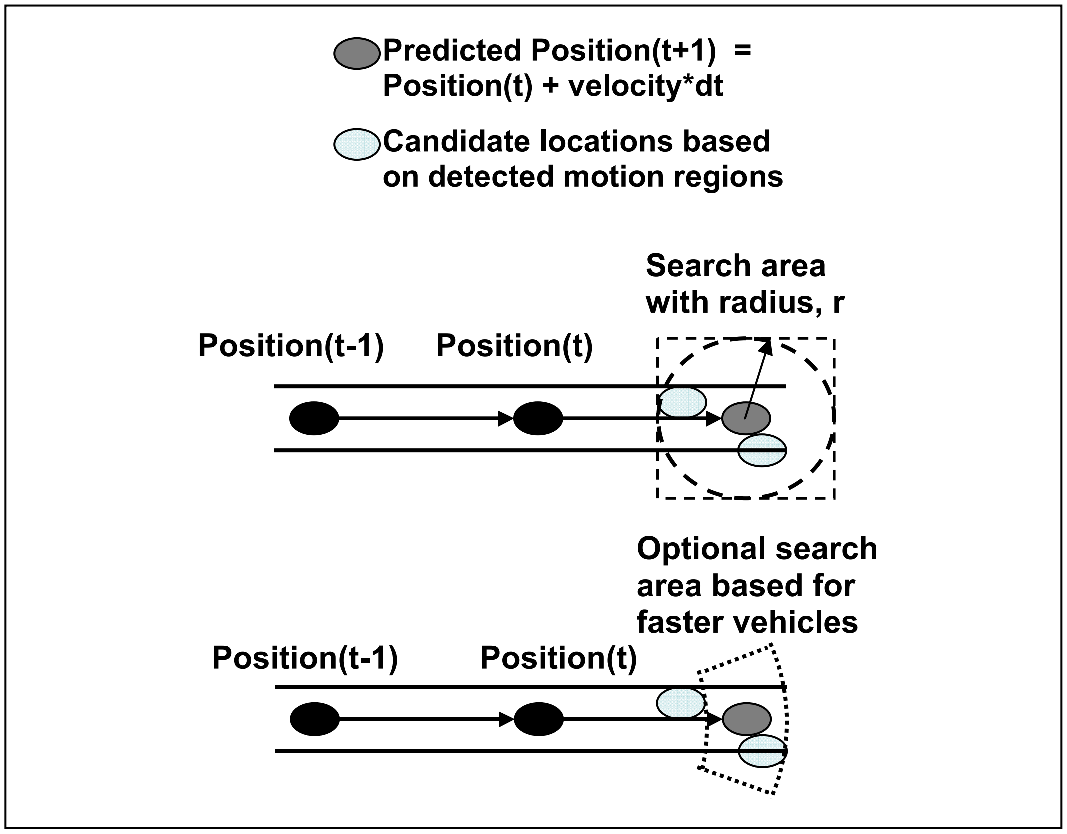

Predicted Position(t+1) = Position(t) + velocity*dt
Candidate locations based on detected motion regions
Search area with radius, r
Position(t-1)
Position(t)
Optional search area based for faster vehicles
Position(t-1)
Position(t)

Figure 5: Diagram of the track association search area depending on vehicle velocity.

An optional ability in the software is to incorporate object information into the binary mover-map used to refine the candidate object locations and for helping in the finding the static object. When turned on, if the vehicle speed is below a certain velocity, the mover map is combined with an object-detection mask. Though optimal object detection routines for the vehicles are still under investigation, the object mask is computed using a thresholded edge-detector together with some morphological operations. This helps keep the full extent of the vehicle detected when it is slowing down and losing pixels in the mover map. The primary disadvantage is that sometimes undesired clutter pixels are included in the mask. An example of a stationary vehicle and its detected object mask are shown in Figure 6.

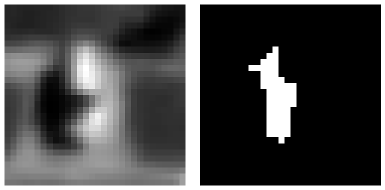


Figure 6: Stationary vehicle and example detected object mask.

Currently four metrics are then computed for each candidate location and the candidate location with the "best" combination of metrics is selected. Track association from frame i to frame i+1 is based on the following dynamics and object appearance features:

- Peak of the phase cross correlation (vehicle shape)
- Average intensity difference (hue difference for color)
- Angle differences (allowable values are function of speed)
- Velocity difference (i.e. acceleration)

These metrics are then normalized and weighted for scoring purposes.

- Phase Cross Correlation (C score) :
  - By default normalized
- Average Intensity difference (I score) :
  - Normalized by taking : 1 – (Intensity difference)/(Maximum intensity difference)
  - Maximum intensity difference is an input parameter (e.g. 60 for 8-bit data)
- Velocity difference (V score) :
  - Normalized by taking : 1 – velocity_diff/maximum_velocity_diff
  - Maximum velocity difference is based on the dynamics model inputs (e.g. maximum allowed accelerations)
- Angle difference (A score)
  - Normalized by taking : 1 - angleDiff/PI (or maxAngleDiff)

Currently, the total score for established movers is by equal weighting.

- Score = (C+I+V+A)/4.0 (for movers)

If the "stop check" option is selected, once the vehicle track has been established for a few frames and the vehicle is moving slow enough, we also check for the static case where the vehicle may have stopped. If the vehicle is not moving, we have only 3 scores to look at. One scoring formula we have tried is the following.

- Score = (C*1.6 + V + I + 0.2)/3.8 (non-movers)

A minimum total score along with a minimum correlation score must be achieved to count as a match. If no match can be found, we allow the vehicle to go on missing status and assign the same velocity and a predicted position to it. If we don't detect the same vehicle within a specific number of frames, the track is terminated.

3.4 Multi-target specific considerations

Tracking multiple or in fact every detected vehicle in an image sequence can be much more complicated than tracking only one vehicle, especially when the traffic density increases and the road network is complex. We can break down the situations into several cases.

- Best case: The vehicle matches exactly one moving or static region in the next frame. We update the track file with this new location and velocity and add one to the track length and keep or set track status to moving or static.
- Two or more vehicles match the same region. This typically occurs when vehicles get very close to each other and their mover regions merge together, or a slow moving large vehicle speeds up. How best to handle this situation is currently under investigation, because the desired outcome depends on the reason for the multiple matching. The code can be toggled to do one of two matching methods:
  - Only allow one match per motion region. (Typically we use this method for speed reasons.)
    - An example of only allowing one match per motion region is shown in Figure 7a and 7b.
  - Allow multiple matches to the same region and allow vehicles to be merged for a maximum number of frames before merging the two track IDs. Proper handling of this case requires knowing that a merge has taken place and keeping track of the appearance of each vehicle separately.
- Mover region starts breaking up.
  - This happens when two or more close together vehicles begin to separate. In this case the best match is chosen to keep the same track ID. The region that didn't get matched will become a newly initialized region with a new track ID. An example is shown in Figure 7c and 7d with track ID 332 breaking up into track ID 332 and ID 431.
  - Another reason this can occur is when a vehicle (this is more common for larger vehicles) slows down and the motion region starts to break up. Often, the front and back of a large vehicle will have separate motion regions when traveling slowly. The degree to which this occurs is dependent on the mover

detection and segmentation algorithm, the spatial sampling interval, how many frames are used, and the frame-rate. It can sometimes be mitigated by using the object detection option described earlier.

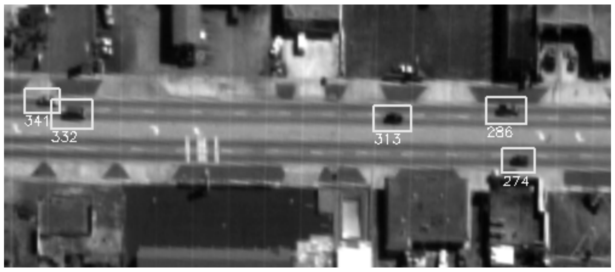


Figure 7a

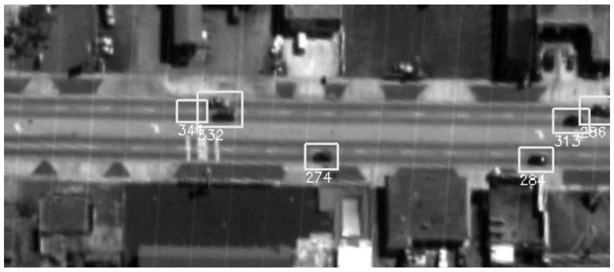


Figure 7b

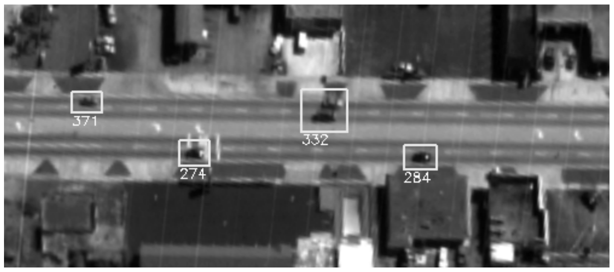


Figure 7c

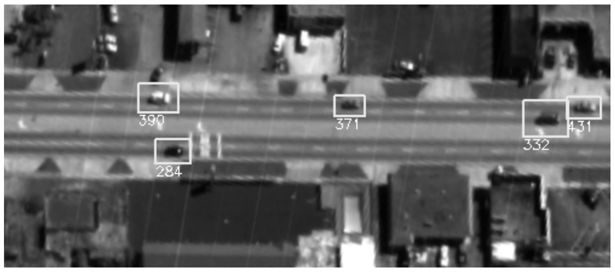


Figure 7d

Figure 7: Example of two vehicles merging together (ID341 and ID332), with the rule that vehicles can't merge. The position ID341 is predicted and still shown in Figure 6b, but is lost several frames later in Figure 6c since the cars (ID 332) are still in close proximity. The now separated vehicle gets a new ID of 431 in Figure 6d.

4. PERFORMANCE ON ULTRA-SCALE IMAGERY

As stated earlier, we are interested in operating on large datasets. In this section we discuss the timing performance of the mover detection and tracking algorithm on datasets from 1k by 1k pixels up to 4k by 4k pixels. While the initial versions of the software were written IDL for ease in prototyping[1], the latest version of the mover detection and tracking libraries are fully implemented in C++ and use Intel performance primitives, math kernel libraries and the threading building blocks[8] to maximize performance. Recall that for many of our applications, real-time is not 30 Hz, but rather on the order of 2 Hz, so we have roughly 0.5 seconds per frame to perform all the operations that we need. If we allow half that time budget for early preprocessing steps, that leaves 0.25 seconds for the tracking.

4.1 Mover detection timing

The temporal median filter at each pixel is a completely parallelizable operation and hence, the parallel_for construct found in the threading building blocks is suitable for this. Timing of the mover detection algorithm with a 5-frame buffer used for the temporal median is shown in Figure 8. A 2 x 3 GHz Quad-Core Intel Xeon Mac Pro was used to generate this plot. From looking at the plot, we can easily stay within our time budget for 1k by 1k video and for 2k x 2k video if multiple CPU's can be utilized. In fact, much higher frame-rates can be supported with smaller image sizes. Note that if only 3-frames are used, the processing time decreases and if a 7-frame buffer is used, the processing time increases roughly in proportion to the total number of pixels processed. In the future, we will be implementing the mover detection on GPU's, which will further enhance performance for this step.

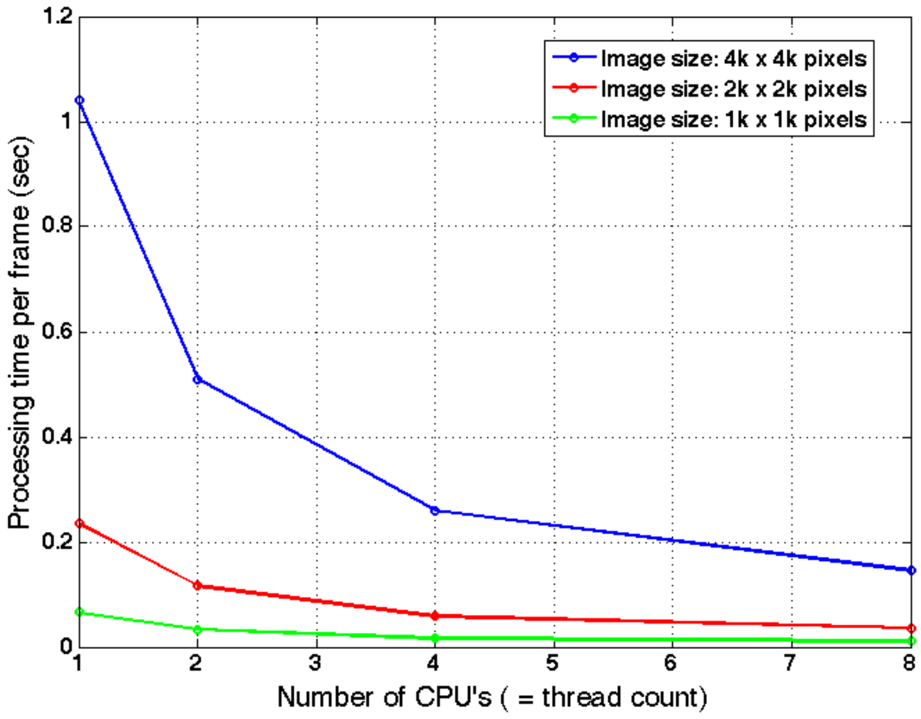


Figure 8: Mover detection performance with a 5-frame buffer up through the binary threshold step using a 2 x 3 GHz Quad-Core Intel Xeon Mac Pro.

4.2 Multi-object tracker timing

There are two ways that the tracking is parallelized. As it turns out many of the Intel library calls are already multi-threaded, so no explicit threading is needed to further increase performance. A further way to parallelize the tracking is to split up the tracked object space. Again, the "parallel_for" construct is used where each thread is responsible for track association of a subset of tracks. The primary drawback to this is that when operating in the mode of a one-to-one track correspondence, some of the ambiguous track associations would not be repeatable if run again on the same dataset due to the random nature of the parallelization. If track ID's 2 and 3 both can match blob 10 with high confidence then whichever thread sees it first will get the track association. Figure 9 and 10 show how the tracker timing scales linearly with the number of objects being tracked. We can also see in Figure 9 and 10 the small effect of parallelizing the loop over the objects being tracked. Because the track association loop only takes a fraction of the full time of the tracker related operations (e.g. checking for new movers), the speedup is not a noticeable boost until many movers are being tracked.

From Figure 9 we see that with many hundreds of objects being tracked in 2k x 2k image streams, the tracking algorithm is well under the time budget of 0.25 sec, even if we must include the CPU-based mover detection step in that budget. In fact for lower object counts, real-time tracking on 4k x 4k image streams is starting to look possible with 8 CPUs, but probably not until the mover detection is ported to the GPU.

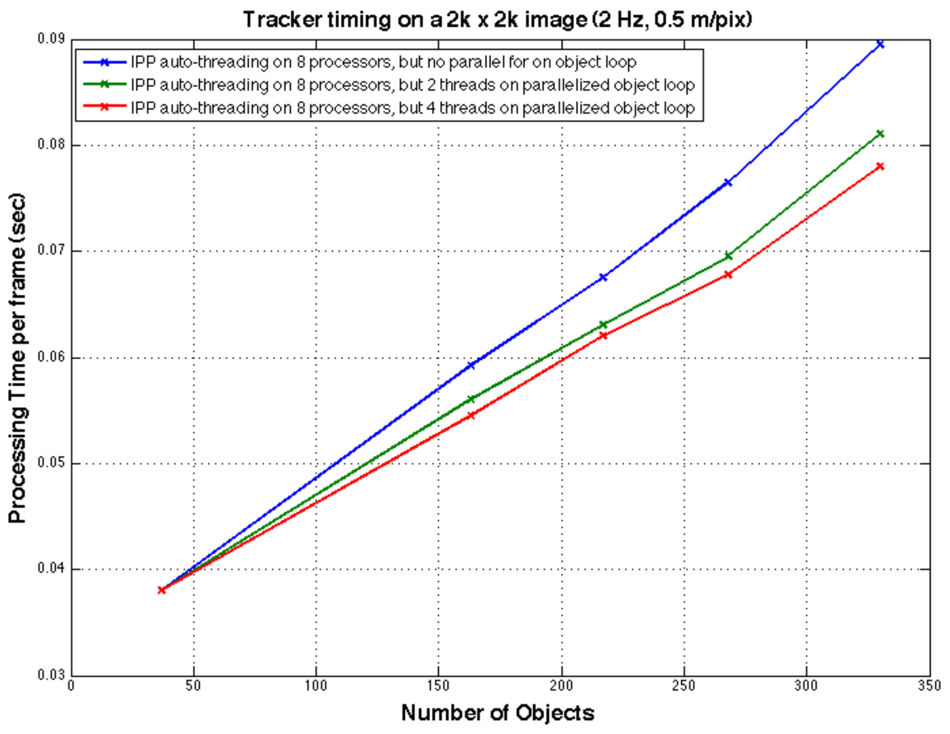


Figure 9: Tracker timing on a 2000 x 2000 pixel image sequence versus the number of objects being tracked

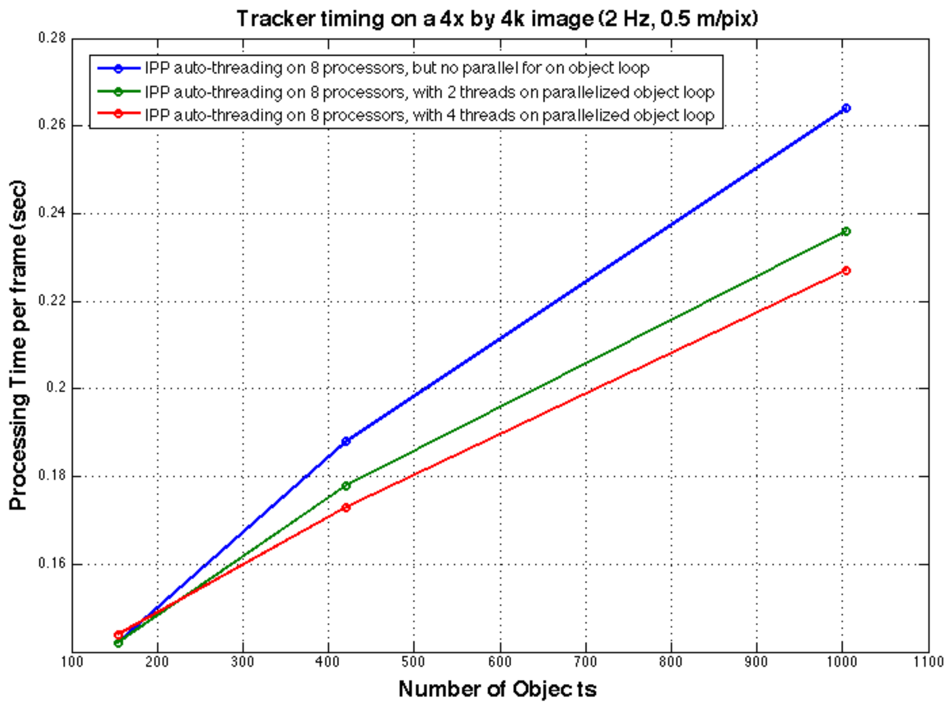


Figure 10: Tracker timing on a 4000 x 4000 pixel image sequence versus the number of objects being tracked

4.3 Tracking Performance

Generally, the performance of the current tracking algorithm is best in the following conditions:

- The vehicles have high contrast with respect to the road
- Vehicles are separated enough from each other or other clutter sources that their motion regions don't merge.
- Clean, stabilized data, but it can handle a fair amount of noise and artifacts.
- Minimal obscurations. We have had much success tracking through tree and building obscurations in situations when the vehicle keeps a constant velocity under the obscuration and does not stop or turn beneath it.
- Vehicle stopping and turning is generally okay.

When such conditions are met, we find the algorithm able to track vehicles for many hundreds of frames (many minutes) without a break in the track. A detailed analysis of the tracking performance of the algorithm is beyond the scope of this paper but will be covered elsewhere. An example of all the tracks greater than ten frames long detected in a ~2 km x 2 km downtown section of San Diego overlaid on Google Earth is shown in Figure 11. The mean track segment length in this complicated urban scene is approximately one minute.

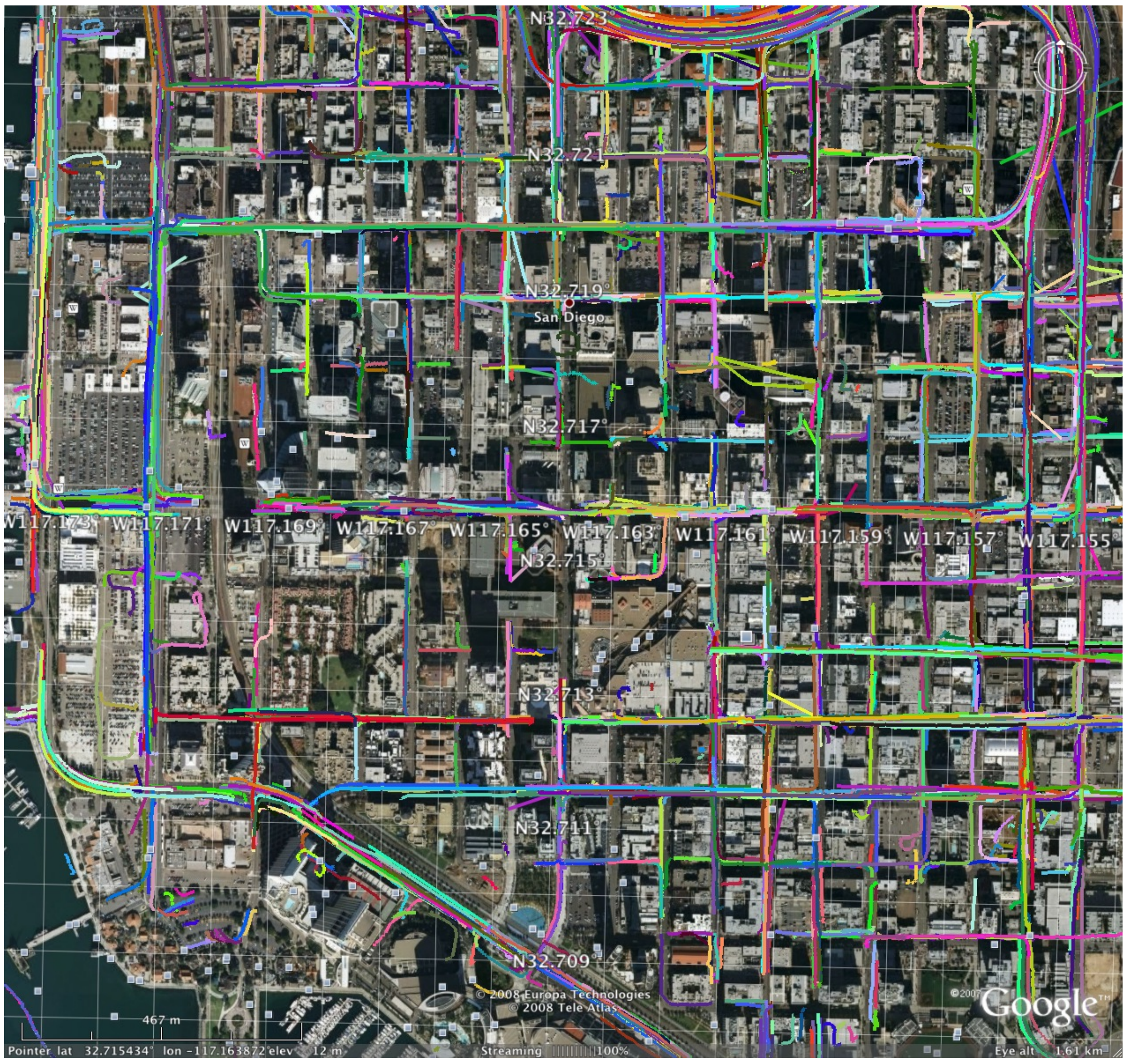

N32.723°
N32.721°
N32.719°
San Diego
N32.717°
W117.173° W117.171° W117.169° W117.167° W117.165° W117.163° W117.161° W117.159° W117.157° W117.155°
N32.715°
N32.713°
N32.711°
N32.709°
© 2008 Europa Technologies
© 2008 Tele Atlas
©2007
Google
467 m
Pointer lat 32.715434° lon -117.163872° elev 12 m
Streaming 100%
Eye alt 1.61 km

Figure 11: Detected tracks from 200 frames (100 seconds worth) of 2 Hz video overlaid all together on Google Earth.

5. SUMMARY

We have developed ultra-scale multi-vehicle detection and tracking algorithms for use on low-resolution and low-frame rate image sequences. We have begun optimizing performance by implementing the algorithms using multi-threaded C++ image processing libraries and parallel constructs. Research will continue in improving the algorithms in terms of tracking accuracy as well as speed.

Acknowledgements: A big thanks goes to Aaron Wegner for performing the image geo-registration and ortho-rectification for some of the datasets used for testing.

6. AUSPICES STATEMENT

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

REFERENCES

[1] C. Carrano, "MTrack 2.0": An ultra-scale tracking algorithm for low-resolution overhead imagery", LLNL-TR-410382, (2009)

[2] M. Kartz, L. Flath, R. Frank, "Real-Time GPS/INS Correlated Geo-Registration and Image Stabilization of Streaming High-Resolution Imagery Utilizing Commercial Graphics Processors ", UCRL-ABS-204226, (2004)

[3] M. Duchaineau, "Progressive Dense Correspondence with Applications to Video Analysis", UCRL-ABS-225824, (2006)

[4] G. Wolberg, Digital Image Warping, IEEE Computer Press, (1990)

[5] S. Cluff, M. Duchaineau, J. D. Cohen and B. Morse , "GPU-Accelerated Hierarchical Dense Correspondence for Real-Time Aerial Video Processing", LLNL-TR-Not-Yet-assigned, (2009)

[6] http://code.google.com/apis/kml/faq.html#whatiskml

[7] http://en.wikipedia.org/wiki/Phase_correlation

[8] http://www.intel.com/cd/software/products/asmo-na/eng/346834.htm#Compilers