LLNL-TR-410382

"MTrack 2.0": An Ultra-Scale Tracking Algorithm for Low-Resolution Overhead Imagery

C. J. Carrano

February 5, 2009

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

"MTrack 2.0": An ultra-scale tracking algorithm for low-resolution overhead imagery

Carmen J. Carrano
Lawrence Livermore National Laboratory, 7000 East Ave, Livermore, CA 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 is very desirable. Typically, this imagery has many thousands of pixels on a side and is characterized by lower resolutions (e.g. ~0.5 meters/pixel to ~2.0 meters/pixel) and lower frame rates (e.g. ~ sub-Hz to several Hz). We describe our ultra-scale capable implementation of a multiple-vehicle tracking algorithm for overhead persistent surveillance imagery. This work builds upon an earlier report[1], where now the algorithm has been modified for improved performance and has been substantially improved to handle much larger datasets in a much shorter time.

1.0 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:

- Vehicles are small (e.g. several to tens of pixels), or low 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 parked cars, sun glints off water or other reflective surfaces, tall building edges, etc.

- Low frame-rates
- Low resolution
- Grayscale information only
- Image stability issues

Our current method 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. 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). One option is to consider this a first stage tracker that generates reliable 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.

2.0 Pre-Processing

The basic processing steps required on raw camera data before tracking is even possible are as follows 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.

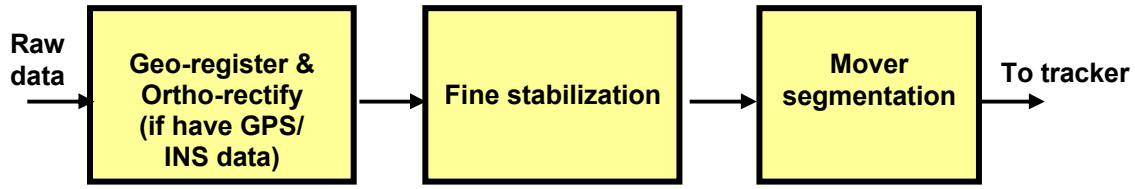


Figure 1: Basic processing steps prior to tracking

2.1 Geo-registration and ortho-rectification

If streamed GPS and inertial navigation system data is available with the image sequence, it is possible to geo-register and resample the data to a NADIR looking view at a specific ground sampling interval. This has the advantage that it is much more straightforward to translate pixel positions in the image to latitude and longitude so that we get can get actual locations and paths of the vehicles for integration with geographical information systems. Describing the algorithm for doing this is 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 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].

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 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 frames, 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 as the frame buffer increases. The other reason is that 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 estimated 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 procedure is shown in Figure 3.

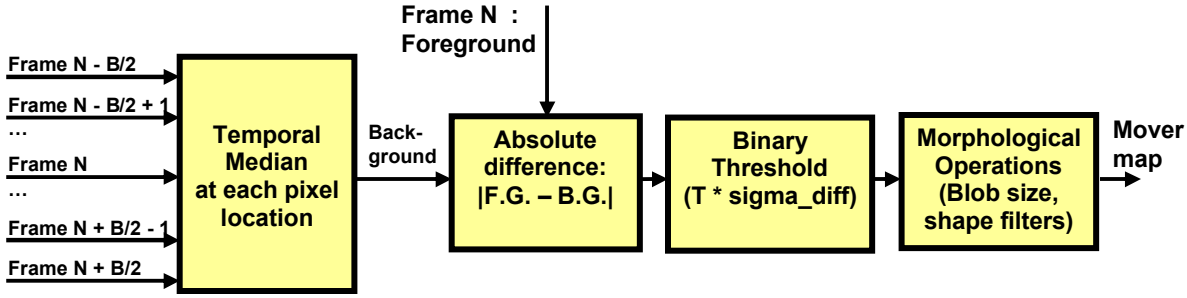
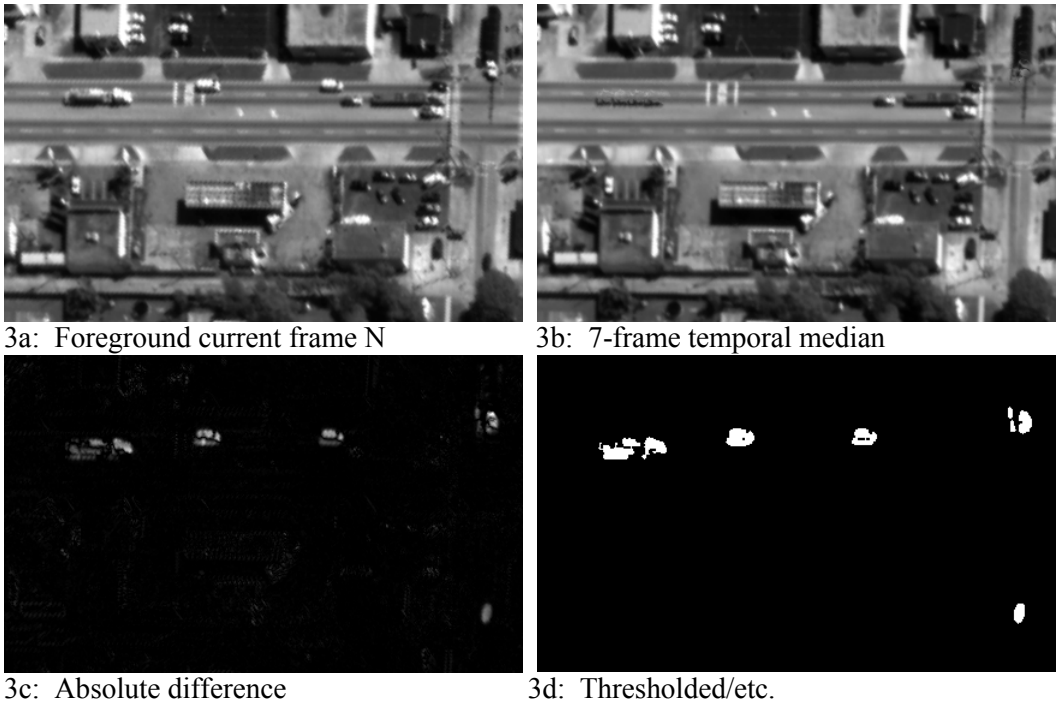


Figure 2: Block diagram of median-filter based motion segmentation.



3a: Foreground current frame N

3b: 7-frame temporal median

3c: Absolute difference

3d: Thresholded/etc.

Figure 3: Example imagery at each step of median-filter based mover segmentation.

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 buildings, 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 straightforward way to deal with movers not associated with vehicles on roads is to mask out regions where vehicles can't go. This mask can be manually or automatically generated from the image itself or knowledge of the road network

3.0 Ultra-scale tracking algorithm

We now describe the current implementation of the multi-vehicle tracker.

3.1 Track file format

In order for the tracking to work, we need to be able to keep a record of each vehicle being tracked as well as some information about that vehicle. Currently, we keep track of the following information in a 2D binary array:

- Frame #
- Track ID
- Center position of region (x, y in pixels)
- Bounding box (x_start, y_start, x_size, y_size)
- X and Y velocity (pixels/frame)
- Track length in # of frames
- Number of frames missed (or # of frames static if in -1 state)
- Track status (0 for just initialized, 1 for moving, -1 for static, -2 for missed)

As the tracking algorithm matures it is possible that we could add additional information or use a separate working track file to store more key features of the tracks or the vehicles themselves (e.g. intensity or color histograms, certainty of the position/velocity estimates, …) We currently don't store lots of information about vehicle appearance because we have its position information and we store a current template of the vehicle as the tracker proceeds.

This track-file format is useful for research purposes only, and is not intended to be used as a final product. There could be multiple final track-file formats whose purpose will be to integrate into further exploitation tools such as track databases and will necessarily include latitude and longitude and date/time-stamps. We have already demonstrated the conversion of the intermediate track data array both to Google Earth KML[5] and to an LLNL in-house XML format for ingestion into a spatial database called E3[6].

3.2 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 position and a bounding box.

For example, the nine regions detected in Figure 3d are initialized as shown in Figure 4. We can see here the importance of getting the mover segmentation right and potential tradeoffs with the threshold and morphological operations. We see here that the big truck on the left is separated into two regions and the big truck on the top-right is separated into three regions. If those trucks were traveling faster, there would be no gaps. If we allow a larger radius in the morphological operation, we can close those gaps, but this comes at the expense of not being able to distinguish two or more vehicles when they are close together because their regions would be merged. Future work may include adding in the extra step of trying to connect motion regions that appear to be from the same vehicle by analyzing object edge or boundary maps or perhaps combining information from using a longer time buffer in the mover detection to better catch the slow movers.

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.

Figure 4: Newly initialized tracks shown with initial bounding boxes and track ID's

3.3 Track association for just initialized 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 and 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 1, a track length of 1, a new position, a new bounding box, and x and y velocities 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. But it can cause confusion when a vehicle slows down causing the motion region to either shrink (on small vehicles) or break up (larger vehicles). The ideal thing to do would be to keep the vehicle area constant once we are sure of its footprint and not let it shrink.

Future work may include fitting an ellipse to the vehicle to detect its orientation thus giving a better idea of where the vehicle could be headed, which the current algorithm does not do. The ellipse fitting should also give better discrimination for track association and rejection of non-vehicles by looking at the ratio of the major to minor axis lengths.

3.4 Track association once moving

Now that we have the initial velocity estimate for each track, we can predict the next position using our constant-velocity dynamics model. 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 maximum radius of the search area is defined at the start of the algorithm based on realistic maximum accelerations and decelerations we would expect from a vehicle. If the vehicle is travelling fast enough, an annular cone is used instead of a circular region. We found it helpful to restrict the search area in this fashion because when a vehicle is going faster it has less ability to maneuver in a given time period, but when going slower, we need to allow for more maneuverability. It is simple to modify the tracker to vary the search area methods or just use one of them. A diagram of these two methods is shown in Figure 5.

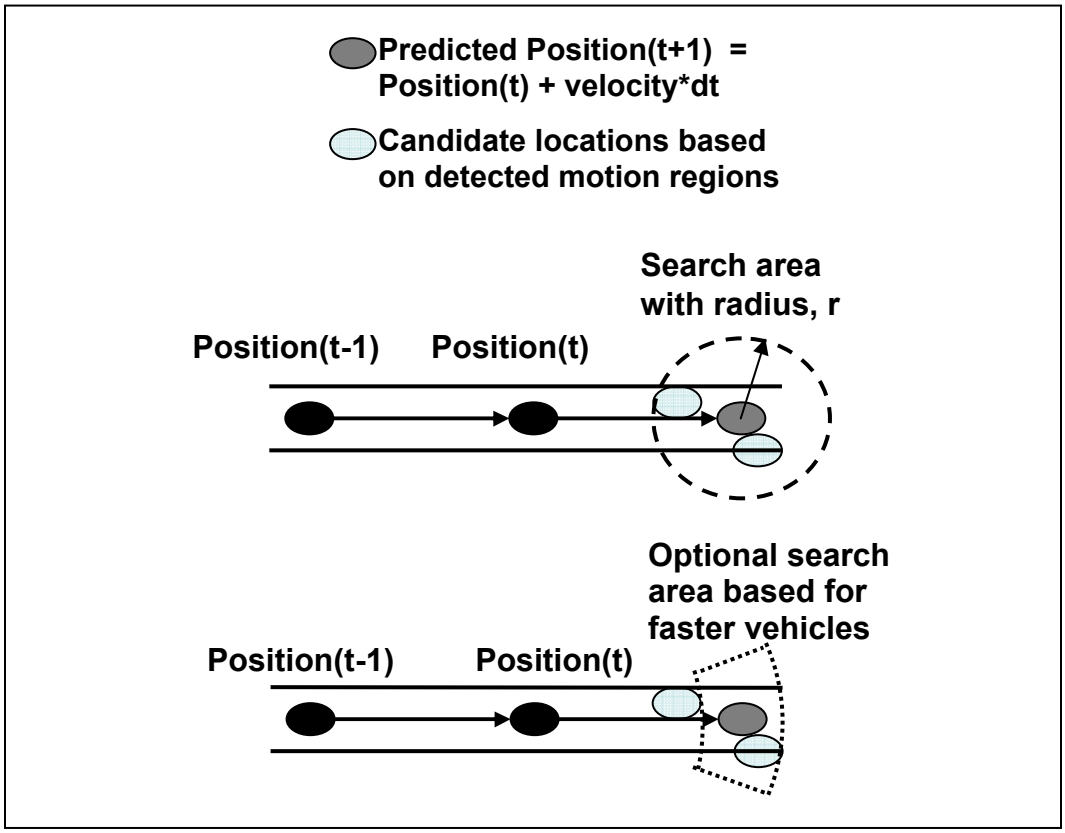


Figure 5: Diagram of the track association search area depending on vehicle velocity.

An optional ability added to this version of 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. If 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. 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_diff/maximum_intensity_diff
    - Maximum intensity 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 diifference 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 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 missing (with a status = -2) and assign the predicted position to it. If we don't detect the same vehicle within a specific number of frames, the track is terminated.

The metrics and scoring combinations chosen here are by no means optimal, but they work satisfactorily in imagery we have performed tracking on. Additional and better metrics are being considered. In particular, a detector that looks in image space to give us a "this really looks like a vehicle" score would be useful. In addition, looking at intensity histograms instead of mean intensity could provide better discrimination for different lighting conditions and multi-colored vehicles. A statistical analysis of the individual scores versus tracking performance could also be done in order to figure out exactly the best score weighting scheme.

3.5 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 1 (moving) or -1 (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 setup to do one of two things:
    - Only allow one match per motion region.
        - 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 can happen when two or more close together vehicles begin to separate. In this case the best match to the larger region is chosen to keep the same track ID. The region that didn't get matched will then become a newly initialized region with a new track ID. An example of this 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 and 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.
    - The same method for handling each case above is used, but ideally we would want to improve how we handle this situation. Options include adding in some additional image processing to detect the vehicle size and shape from the imagery or perhaps from the mover map at an earlier detection, when we know the size estimate will be accurate. For instance, when the large vehicle is moving fast enough, the mover map size estimate will be most accurate. In either case, once we know the estimated true size we want conserve the vehicle area and not let it break up.

Figure 7a

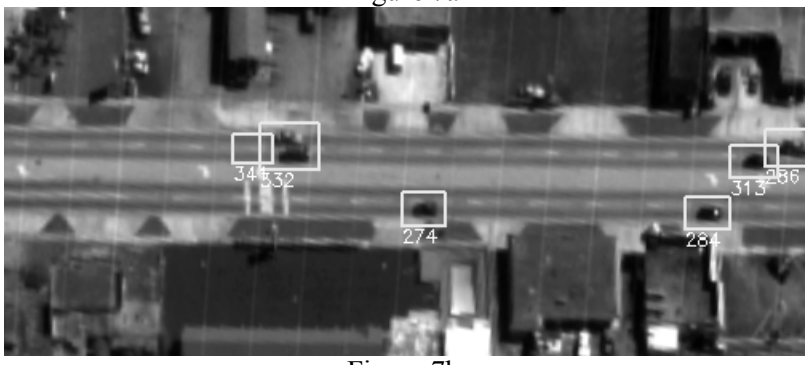


Figure 7b

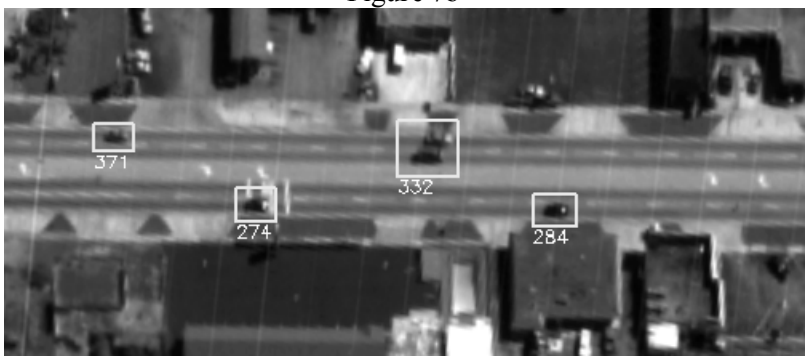


Figure 7c

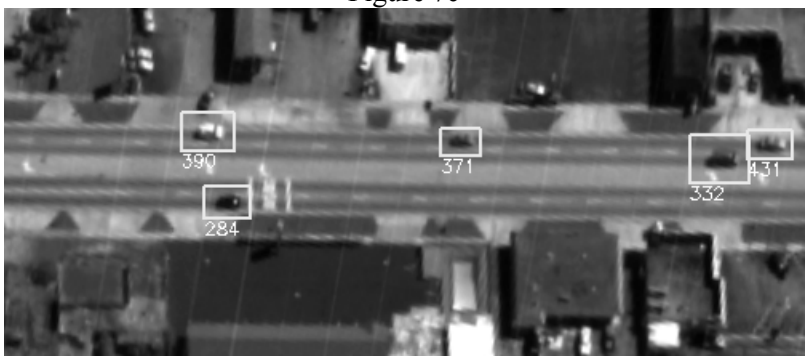


Figure 7d

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.

3.6 Track filters

After the tracking for the sequence is completed and a track-file generated, we find it useful to remove certain tracks that will probably be of little interest. The filters used most often are the following.

- Track-length filter. User specifies the minimum length of the track in number of frames. All tracks less than that length are eliminated.
- Missing-track filter. This is useful to do to the track file prior to creating a movie. It removes all entries from the track-file with a missing status (= -2).

4.0 Ultra-scale data considerations

In testing our (IDL based) algorithm out on large datasets (4000 x 4000 pixels by 1000 frames) we found a number of bottlenecks that needed to be addressed in order to track 100's to 1000's of vehicles simultaneously per frame in a timely fashion. Bottlenecks included:

- Data I/O – Since we can't just read an entire dataset into physical memory and process it all at once, we needed to adjust the algorithm so that it only reads in a single frame a time and keeps only two frames worth of data in memory.
    - Earlier versions were reading in the templates for matching from much earlier frames, (in case there was an obscuration) but now we keep small template images of each object from past frames in memory with no need to go back and read in data from earlier frames.
- Minimize computations on the full size image. Try to perform operations in local regions as needed.
- Track data management – With the possibility for seemingly endless track data arrays with tens of thousands to millions of tracks, we have implemented a circular buffer approach to the track data. Only the most recent track data is kept in memory, (e.g. 1 minute worth), while the older track data is streamed to disk.
- Migrating to C code. IDL is great for prototyping algorithms, but not necessarily intended for real-time performance. Fortunately with IDL, when certain routines are found to be too slow, they can be rewritten in C and called from IDL to gain additional performance. (e.g. Speedup of 100 was gained on the routine that assigns all the mover region statistics to an array.)
    - The entire processing pipeline will eventually be implemented in C/C++ with multi-threading to achieve "real-time" performance for up to a certain image size and vehicle count.
    - "Real-time" depends on the camera frame rate. (2 Hz, 10 Hz, 30 Hz, etc.)

Current tracker timing results on a 3 GHz Mac Pro are as given below.

Test #1 with ground sample distance (GSD) at ~0.5 m/pixel and including disk I/O:

Data Size	Object Count (avg. per frame)	Total Time	Time per frame (sec)
4000x4000x100	1200	3.5 min	2.1
4000x4000x500	1533	20 min	2.4
4000x4000x1000	1761	46 min	2.8
4000x4000x192	174	4.0 min	1.3
4000x4000x192	424	5.9 min	1.8
2000x2000x180	68	45.0 sec	0.25

2000x2000x180	176	62.7 sec	0.35
1000x1000x200	36	17.4 sec	0.087
1000x1000x200	94	30 sec	0.15

Test #2: Read the image data and mover map all into memory first:

DataSize	Object Count (avg. per frame)	Total Time	Time per frame (sec)
250x250x100	5	0.83 sec	.0083
250x250x200	4.5	1.5 sec	.0075
250x250x200	2.3	0.85 sec	.0043
500x500x50	22	3.3 sec	.066
500x500x100	26	7.9 sec	.079

These tests show that for both low and high object counts that the timing scales with the number of frames, as it should. The scaling isn't exact, because the average object counts are not exactly constant over time. The tests also show how the timing scales with the number of objects, which scales with a slope much less than one for the disk-bound case and much closer to one for the in-memory case.

5.0 Tracking Performance

Generally, the performance of the "Mtrack 2.0" 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. This is an active research area.
- Clean, stabilized data, but it can handle quite a bit 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 is 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. Detailed analysis of the tracking performance of the algorithm will be covered elsewhere.

An example of tracks detected in a ~2 km x 2 km downtown section San Diego overlaid on Google Earth is shown in Figure 1.

6.0 Conclusions

We are in the process of developing and implemented ultra-scale multi-vehicle tracking algorithms and software for persistent surveillance imagery. This version, "Mtrack 2.0", is written in IDL with several C function calls. It has been optimized to handle large spatial and long temporal image sequences. Future reports may include various improvements to the tracking software, both in speed and algorithm details.

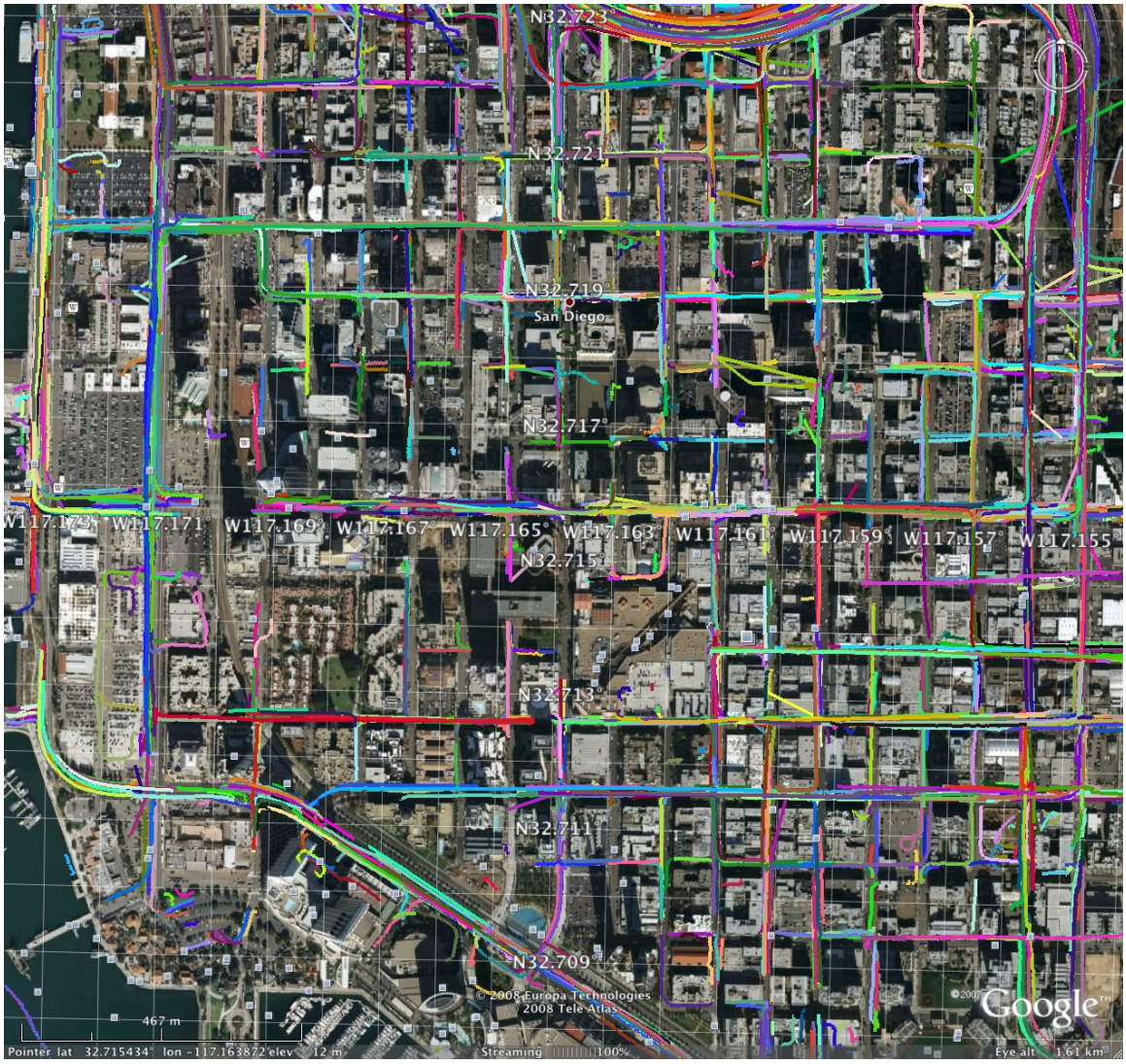


Figure 8: Detected tracks from 200 frames (100 seconds worth) of 2 Hz video overlaid all together on Google Earth.

References

1. C. Carrano, "Mtrack 1.0: A multi-vehicle, deterministic tracking algorithm", June 2008, LLNL-TR-404638
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
3. M. Duchaineau, **"**Progressive Dense Correspondence with Applications to Video Analysis", UCRL-ABS-225824
4. G. Wolberg, Digital Image Warping, IEEE Computer Press, 1990
5. **http://code.google.com/apis/kml/faq.html#whatiskml**
6. **https://www-gs.llnl.gov/E3_GUXS2.xsd,** UCRL-MI-232806
7. http://en.wikipedia.org/wiki/Phase_correlation