LLNL-TR-400667

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

"Eztrack": A single-vehicle deterministic tracking algorithm

C. J. Carrano

January 23, 2008

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

"Eztrack": A single-vehicle deterministic tracking algorithm

C. Carrano, LLNL

June 6, 2007

1.0 Introduction

A variety of surveillance operations require the ability to track vehicles over a long period of time using sequences of images taken from a camera mounted on an airborne or similar platform. In order to be able to see and track a vehicle for any length of time, either a persistent surveillance imager is needed that can image wide fields of view over a long time-span or a highly maneuverable smaller field-of-view imager is needed that can follow the vehicle of interest. The algorithm described here was designed for the persistence surveillance case.

In turns out that most vehicle tracking algorithms described in the literature[1,2,3,4] are designed for higher frame rates (> 5 FPS) and relatively short ground sampling distances (GSD) and resolutions (~ few cm to a couple tens of cm). But for our datasets, we are restricted to lower resolutions and GSD's (≥0.5 m) and limited frame-rates (≤2.0 Hz). As a consequence, we designed our own simple approach in IDL which is a deterministic, motion-guided object tracker. The object tracking relies both on object features and path dynamics. The algorithm certainly has room for future improvements, but we have found it to be a useful tool in evaluating effects of frame-rate, resolution/GSD, and spectral content (eg. grayscale vs. color imaging). A block diagram of the tracking approach is given in Figure 1. We describe each of the blocks of the diagram in the upcoming sections.

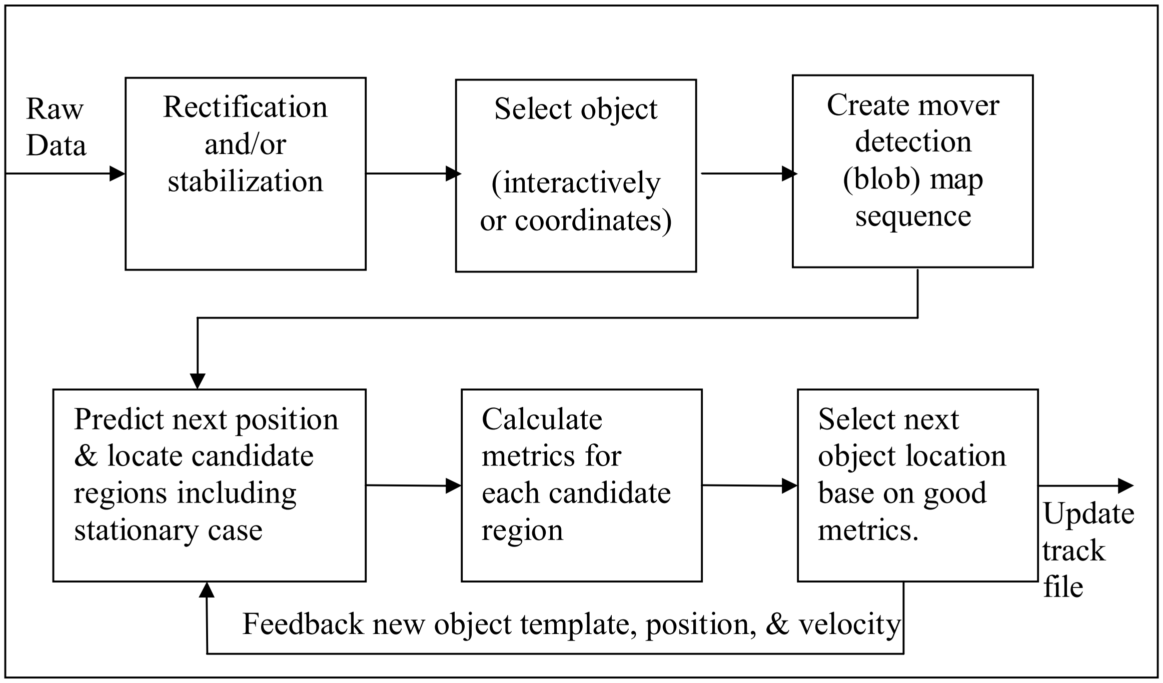

Raw Data
Rectification and/or stabilization
Select object (interactively or coordinates)
Create mover detection (blob) map sequence
Predict next position & locate candidate regions including stationary case
Calculate metrics for each candidate region
Select next object location base on good metrics.
Update track file
Feedback new object template, position, & velocity

Figure 1: Block diagram of tracking approach.

2.0 Pre-processing: Stabilization

The tracker treats the image sequence as if it were acquired from a stationary platform. As a result, it is necessary for the data to be stabilized from frame to frame. For much of the data available to us, it was both geo-registered[5] and then stabilized with a translation, rotation, and scale sensitive stabilization algorithm[6] to remove small geo-registration errors. In some other datasets where geo-registration was not possible we performed affine image registration[7] to register all the frames of the sequence to a reference frame.

3.0 Vehicle Selection

Before tracking can begin, it is necessary to select the object to be tracked, either automatically or manually. Rather than automatically try to detect multiple vehicles and follow them all, the tracker simply requires that the pixel coordinates of a bounding box around the vehicle be supplied. The software allows the user to input them directly or draw them interactively. When using fast frame rates, it is sufficient to perform the vehicle initialization with a single frame, because the vehicle will not move very many pixels to the next frame. But with slower frame rates, it is necessary to initialize the tracker with boxes around the same vehicle in the first and second frames. This allows the initial velocity to be known and used in the prediction of the location in the next frame. Figure 2 shows a box drawn around the vehicle to be tracked. Future plans include working on the track initiation problem.

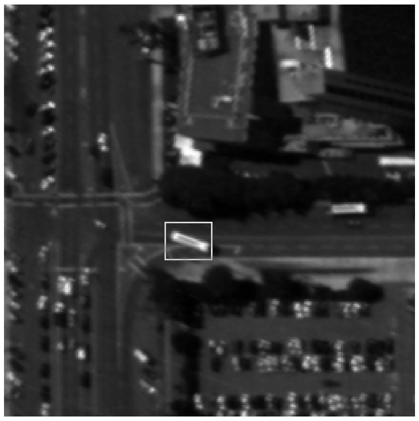


Figure 2: Box drawn around the target to be tracked

4.0 Mover detection

There are many motion detection algorithms described in the literature[8][9] each with its own strengths and weaknesses. Eztrack was written in a modular fashion so that any mover detection method can be tested or used. Since the data is always pre-stabilized before tracking, contains small movers, and has slow frame-rates, we chose three-frame differencing[1] for the initial default mover detection algorithm. It is fast for large datasets and can easily be adapted to multi-spectral data. The three-frame differencing algorithm is quite simple: A pixel is chosen to be moving if:

$$\left|I_{n+1}(x) - I_n(x)\right| > T_{n1}(x) \quad \text{AND} \quad \left|I_n(x) - I_{n-1}(x)\right| > T_{n2}(x) \qquad [1]$$

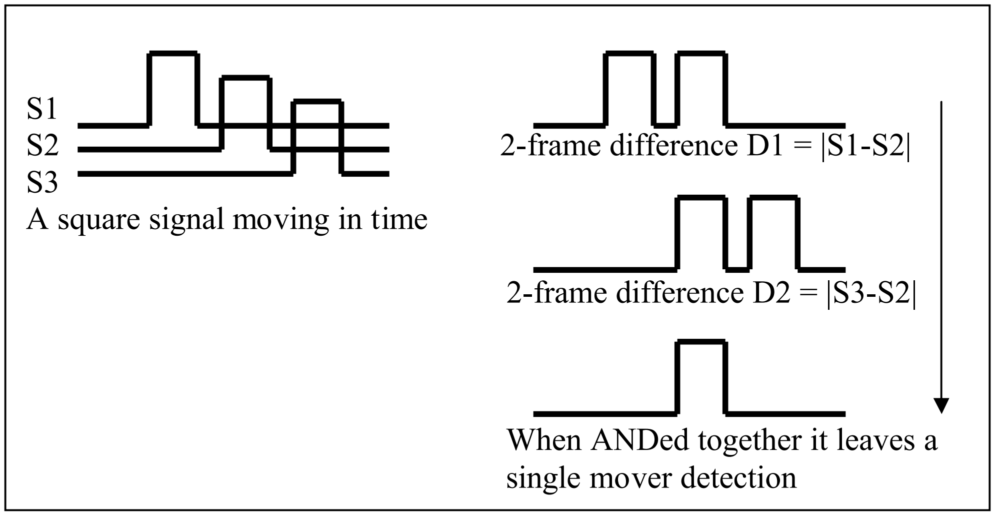

S1
S2
S3
A square signal moving in time
2-frame difference D1 = |S1-S2|
2-frame difference D2 = |S3-S2|
When ANDed together it leaves a single mover detection

Figure 3: Illustration of three-frame differencing in 1-D

Where $I_n(x)$ is the intensity of the pixel at spatial coordinate x for frame number n, and $T_n(x)$ is a threshold defining a statistically significant intensity change at each pixel location which we currently define as a percentage of the maximum intensity change (i.e. max(ΔI) * scale factor.) This scale factor is allowed as an input to the mover detection routine. Typical values for this scale factor are 5% to 15%. We also allow the option to have the threshold vary over the image or let it be a constant value, though we have not exercised the variable threshold option yet.

If the imagery is multi-channel, we have several choices. We can form a grayscale intensity image from the multiple channels and proceed as before. Or we can try something that takes advantage of the multiple channels, such as the popular Minkowski metric[11], replacing the intensity differences from Equation 1 with:

$$\left(\sum_{k=1}^{m}\left|I_{k,i+1}(x) - I_{k,i}(x)\right|^{p}\right)^{1/p} > T_n(x)... \qquad [2]$$

Where k is the channel number (e.g. color - R, G, B), i is the frame number and x is over space. If $p = 2$, equation 2 is a Euclidian distance metric. We have found good results with $p=3$ as well.

Once the initial mover map is created, it is necessary to remove very small detections and perhaps very large detections. A blob region-size filter is applied to keep only mover blobs greater and/or less than a certain size. Typical minimum blob sizes are 3 to 5 pixels, but it can vary depending on image resolution and the size of the objects of interest. The effect of the threshold and minimum blob size is shown in Figure 4. It is possible to apply any number of image segmentation routines to the mover map to either enhance movers of interest or reduce uninteresting or false movers. Future work will include investigation into this area.

Once the final mover map is created, it is then used to guide the tracker.

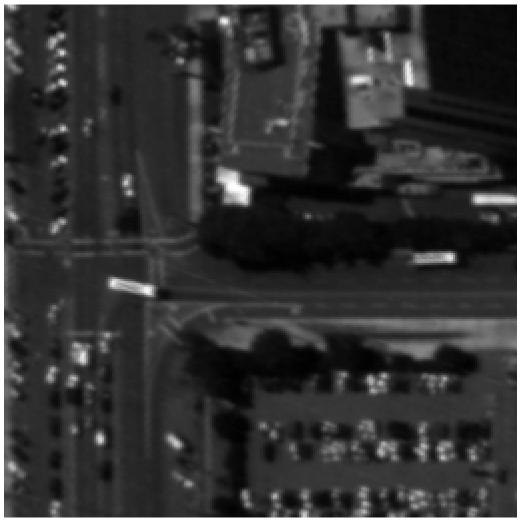
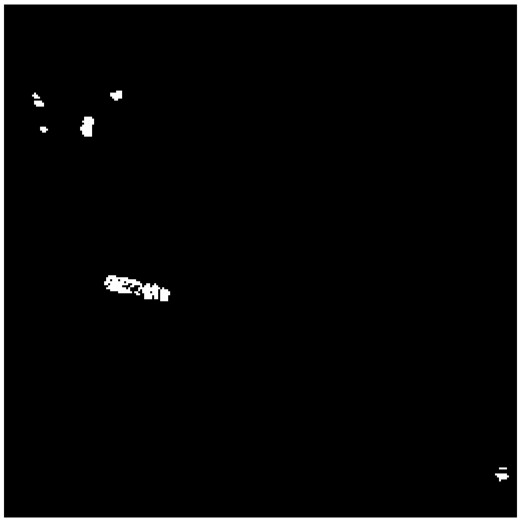
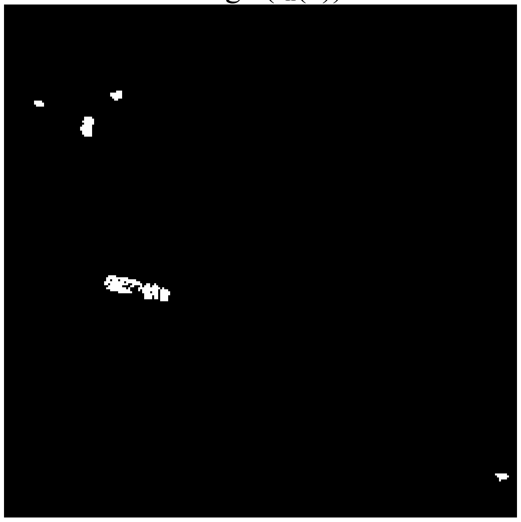
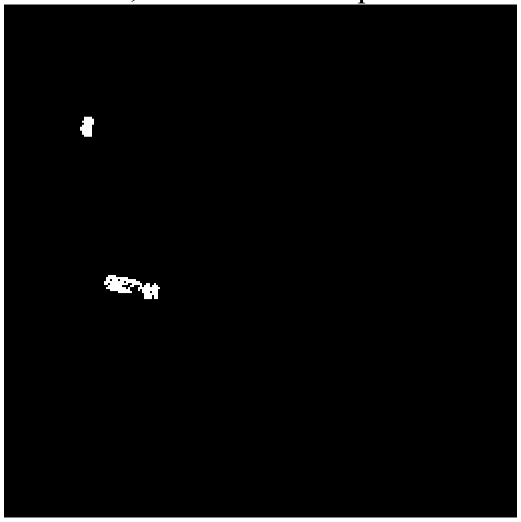
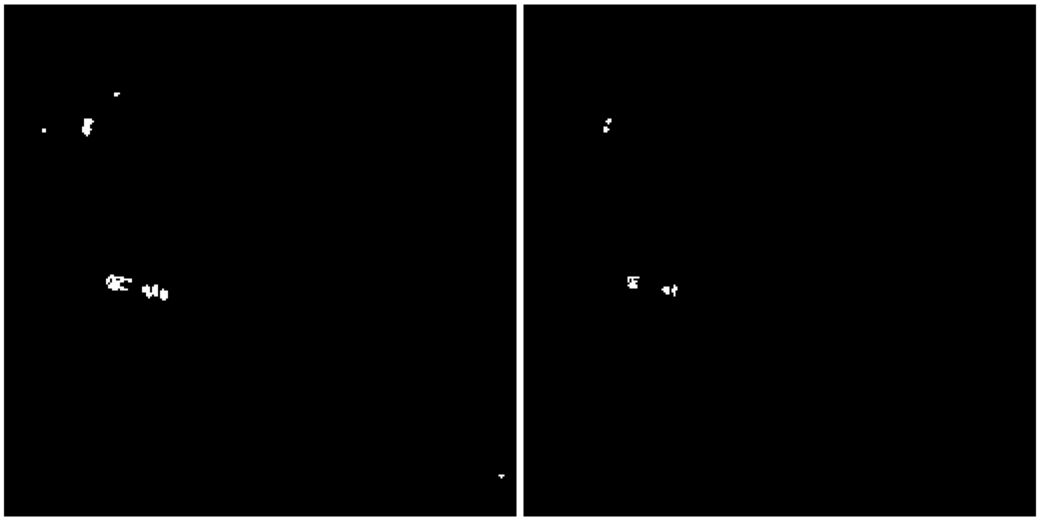


Image ($I_n(x)$)


T= 5%, MinBlobSize=3 pixels


T = 5%, MinBlobSize=10 pixels


T = 5%, MinBlobSize= 50 pixels



T = 10%, MinBlobSize=3 pixels T=20 %, MinBlobSize= 3 pixels

Figure 4: Effect of Threshold and blob region size filter on the mover map.

5.0 Tracking

After getting the initial position, optional initial velocity estimate, bounding box of the object (section 3.0), and the mover map for each frame (section 4.0), the tracker loops over the frames. At each frame it predicts the position of the object based on the previous position and the last known velocity (i.e. $p_{n+1} = p_n + v_n \Delta t$)

Within a user-defined radius (e.g. typically 10 to 25 pixels) around the predicted position, it searches for candidate blobs in the mover map, including the stationary case which is not part of the mover map. The candidate blobs are identified and labeled by multiplying the current mover map frame by a binary circular mask with its center at the predicted position and with a radius of the maximum position deviation allowed. A segmentation routine such as IDL's label_region.pro is used to label each blob for the search. It then loops through those candidate positions, including the stationary case, and computes several quantities to determine where it thinks the object moved to (or didn't move to). The tracker currently uses four basic metrics to determine the object location in the next frame:

- Cross correlation
  - The peak of the cross-correlation between the mean-removed, hanning-windowed object template and candidate object. (In some cases we tested the cross correlation of the Laplacian of the objects which was helpful sometimes when resolution was reduced or the object was small.) A running average value of the maximum correlation value over time is also kept.
- Path-coherence (Φ) [10]
  - Checks to make sure that the direction and velocity changes in consecutive images are smooth.

- $\Phi(P_{n-1}, P_n, P_{n+1}) = w_1(1-\cos\theta) + w_2(1 - 2\frac{\sqrt{s_k s_{k+1}}}{s_k + s_{k+1}})$, where the variables are described in Figure 5. We found it suitable to equally weight $w_1 = w_2 = 0.5$. A value near zero means very good path coherence (e.g. < 0.2)

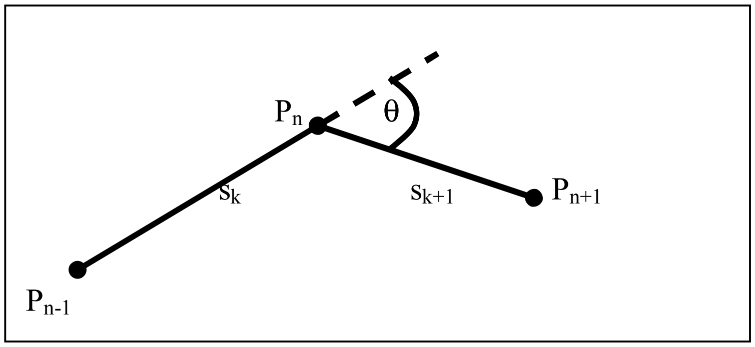


Figure 5: Path coherence, definition of the angle θ and the distances s_k and s_{k+1}. The P_n's are the center positions of the objects.

- Velocity difference
  - A simple difference of the potential new velocity from the previous velocity. (In pixels per frame, but could translate to a real velocity such as meters/sec). A maximum acceptable velocity difference is set in the beginning of the code and can be set by the user. Typical values used for our data was 10 to 25 pixels/frame.
  - We have this redundant metric because we found that path coherence alone was not sufficient to make the best choice.
- Average intensity difference for grayscale imagery, average hue difference for color imagery, or a minimum matched filter threshold for multi-band data.
  - The average intensity, hue, or spectral matched filter is calculated over the moving object regions to be checked in the current frame and compared to that of the previous frame. An overall threshold has to be met for that candidate region to be considered at all and small intensity or hue differences are given stronger weighting in choosing the next position of the object.

A somewhat complicated set of checks and cross-checks is performed on each of the four values to determine the next position of the object. These checks can be easily modified to fit the nature of the data if need be. For example, we could put more emphasis on the correlation for higher resolution data or put more emphasis on the path dynamics for lower resolution data. In general, we are looking to find the object that has the minimum average intensity (or hue) difference, highest correlation, best path coherence and least velocity difference. But often, that is not the case and we find one or two of the values will be second or third choice. So the trick is in how much weight to give to each metric and in what situations. Sometimes the correlation will be high, but the path coherence or velocity difference will be bad, so the algorithm may not pick that target if it finds another object that has a better path coherence and velocity difference so long as the correlation is higher than some threshold (a percentage of the maximum correlation found). The logic of one grayscale version of the algorithm is similar to the following:

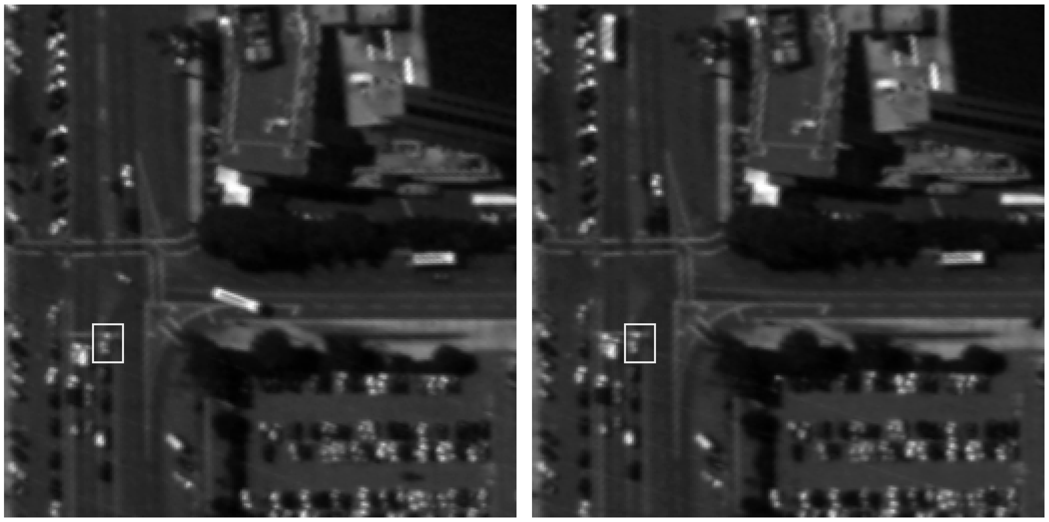
```
If (IntensityDiff < Thresh(e.g. 60) )
{
   If (PeakCorr > MaxPeakCorr) OR
      (PathCoh<0.25 AND PeakCorr>0.5*AvgCorr AND VelDif<MinVelDif) OR
      (PathCoh<0.05 AND PeakCorr>0.2*AvgCorr AND VelDif<MinVelDif*0.25)
   {
      If (VelDif < MinVelDif AND PathCoh<MinPathCoh ) OR
         {VelDif < MinVelDif AND PeakCorr>MaxPeakCorr)
      {

            Update MaxPeakCorr
            Update MinPathCoh
            Update MinVelDif
            Update Mover choice
      }
   }
}
```

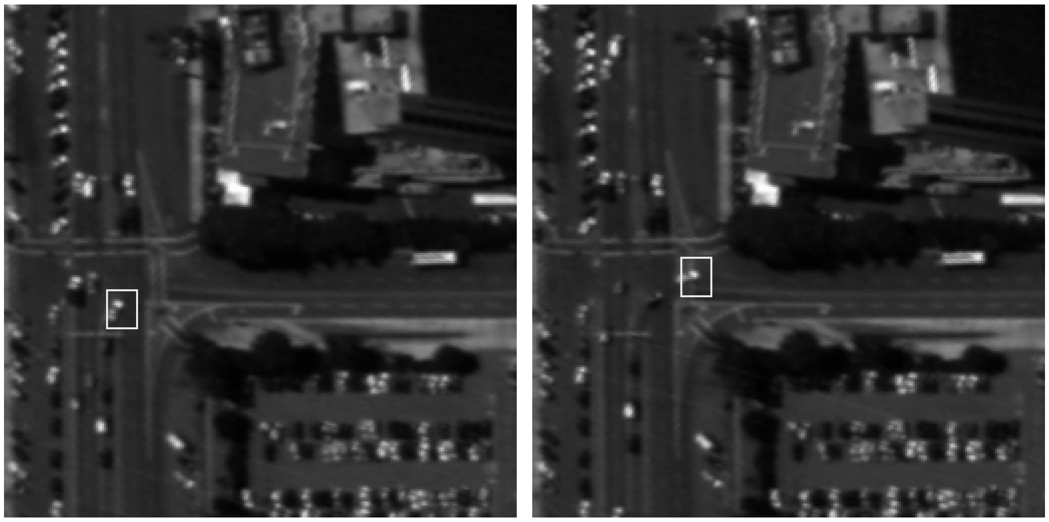
If no objects can be found with high enough correlation or good enough path dynamics, then the tracker indicates that there was no match and will assume the target became occluded and predict the location of the object in the next frame and search again in the frame after that. The prediction simply sets $p_{n+1} = p_n + v_n \Delta t,$ and it does not update the template of the object in that case. If after a pre-defined number of frames (e.g. 2 to 5) the object is not picked up again, it assumes the object is lost and stops trying to track it.

A text file with the object position (center), bounding boxes, and velocities is saved for future use such as displaying multiple tracks back on the data. The object template sequences can also be saved.

Figure 6 shows sample frames of the detected track overlaid on a stationary car that begins to move and turn a corner. The GSD is 0.5 m and the frame-rate is 2.0 Hz.



Initial frame with user drawn box   Detected car location after 30 frames



Detected location after 40 frame   Detected location after 45 frames

Figure 6: Tracking example where a car is stationary for 30 frames then starts to move and turn a corner.

Summary and Conclusions

The optimal metrics and how best to use them in tracking vehicles or other objects is still under investigation, but the current tracker implementation works satisfactorily (>70-80% success rate) on many of the datasets we have access to with variable resolutions and frame-rates. Of course, it works best in uncluttered environments with distinct looking vehicles (Resolution/GSD < 1 m/pixel) and at better frame rates (>1 Hz), but we have demonstrated good performance at the 0.5 Hz rates depending on resolution and traffic density. Future work on this tracker will include expanding it to multiple vehicles, alternative mover detection schemes, allowing the bounding box to change size, and continued work on best choice and use of the track feature metrics.

References

[1] R. T. Collins, et. al. "A System for Video Surveillance and Monitoring", VSAM Final Report, Carnegie Mellon University, 2000
http://www.cs.cmu.edu/~vsam/research.html

[2] A. J. Lipton, H. Fujiyoshi, R. S. Patil, "Moving target classification and tracking from real-time video", Applications of Computer Vision , 1998. WACV '98. Proceedings., 4th IEEE workshop on, Pg 8-14, (1998)

[3] W. Bell, P. Felzenszwalb, D. Huttenlocher, "Detection and long term tracking of moving objects in aerial video", Technical Report, Computer Science, Cornell, March 1999
http://www.cs.cornell.edu/vision/wbell/identtracker/

[4] G. Baldini, et. al. “A simple and robust method for moving target tracking”, SPPRA 2002

[5] 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

[6] B.S. Reddy and B.N. Chatterji,”An FFT-Based Technique for Translation, Rotation, and Scale-Invariant Image Registration”, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 5(8), pp. 1266-1271, 1996

[7] M. Duchaineau, **“**Progressive Dense Correspondence with Applications to Video Analysis”, UCRL-ABS-225824

[8] M. Piccardi, “Background subtraction techniques: a review”, The ARC Center of Excellence for Autonomous Systems, Faculty of Engineering, UTS, April 15, 2004 www-staff.it.uts.edu.au/~massimo/BackgroundSubtractionReview-Piccardi.pdf

[9] S. Cheung, C. Kamath, “Robust techniques for background subtraction in urban traffic video”, Proceedings of the SPIE, Volume 5308, pp. 881-892 (2004).

[10] M. Sonka, V. Hlavac, R. Boyle, “Image Processing, Analysis, and Machine Vision” 2nd ed., PWS Publishing, pp. 700-702 (1999) ISBN 0-534-95393-X

[11] C. C. Chibelushi, Presentation : “Statistical Pattern Recognition, Part 1: Clustering”, Staffordshire University, 2007
www.soc.staffs.ac.uk/ccc1/IPCVPR/IPCVPRpatrecPI04.ppt

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.