# An Open Source Tracking Testbed and Evaluation Web Site

**Robert Collins, Xuhui Zhou, Seng Keat Teh**
**Robotics Institute, Carnegie Mellon University**
**[rcollins, xuhui, steh]@cs.cmu.edu**

## Abstract:

We have implemented a GUI-based tracking testbed system in C and Intel OpenCV, running within the Microsoft Windows environment. The motivation for the testbed is to run and log tracking experiments in near real-time. The testbed allows a tracking experiment to be repeated from the same starting state using different tracking algorithms and parameter settings, thereby facilitating comparison of algorithms. We have also developed a tracking evaluation web site to encourage third-party self-evaluation of state-of-the-art tracking algorithms. The site hosts source code for the tracking testbed, a set of ground-truth datasets, and a method for on-line evaluation of uploaded tracking results.

## 1. Introduction

This work addresses the need for tools and community resources for evaluating and comparing the performance of tracking algorithms. The domain of current interest is tracking ground vehicles from airborne sensor platforms. To facilitate evaluation of tracking algorithms on such datasets, we have implemented an open source, interactive tracking testbed using C and Intel OpenCV routines [3] within the Microsoft Windows operating environment. The testbed allows a tracking experiment to be "replayed" using different tracking algorithms and parameter settings.

We have also set up a tracking evaluation web site where researchers can self-evaluate their own algorithms. The primary features of the site are: 1) availability of data sets with ground-truth results; 2) sample baseline tracking algorithms implemented within the tracking testbed; 3) provisions for uploading results in a standard data format; 4) a mechanism for on-line, automated scoring of uploaded tracking results; 5) a table of algorithm rankings and pointers to publications describing each algorithm.

**Related Work**

The tracking evaluation web site is modeled after the Middlebury stereo evaluation web page [8]. This stereo page has been quite successful, since it is now nearly impossible to publish a new algorithm for two-frame stereo matching without documenting how well the algorithm performs on the Middlebury datasets. We believe that much of the success of the Middlebury site is due to the competitive nature of the online algorithm rankings table. The competitive nature of the Middlebury page makes it fun for people to participate in the evaluation.

A web site for evaluation of face recognition algorithms is available from Colorado State University [1]. This site contains baseline implementations of popular face recognition algorithms and a set of recommended evaluation protocols. It does not currently contain an on-line algorithm ranking system.

Evaluation of tracking systems is addressed annually by the IEEE Performance and Evaluation of Tracking and Surveillance (PETS) workshop series [6]. The goal of the PETS workshops is for researchers to run their tracking algorithms on the same datasets, submit results in an xml format, and publish their algorithm and results in the proceedings. Most of the PETS datasets and algorithms are geared towards stationary camera surveillance, and therefore most algorithms rely on statistical background subtraction techniques to detect moving objects – an approach that is not directly relevant to airborne datasets where the camera is constantly in motion.

In Section 2 we provide an overview of the open source tracking testbed, and describe in detail one module available within it that provides background motion estimation and foreground motion prediction.
Section 3 presents the tracking evaluation web site, with an emphasis on its scoring metrics.

## 2. Open Source Tracking Testbed

### 2.1 Tracking Testbed System

We have implemented a GUI-based tracking testbed system in C and Intel OpenCV, running within the Microsoft Windows environment. We are distributing the code for public download as an open source resource. A compiled executable is available for people who just want to run the tracker as is. Complete source code is available for people who want to modify the functionality, or add their own tracking algorithms into the testbed. A user manual and programmer reference guide are included with the distribution.

Figure 1 shows a screen capture of the tracking testbed. The object being tracked is outlined with a blue bounding box. Along the top of the GUI is a menu of options. At the far left, the "Open" option opens a directory browser that allows the user to select the first file (frame) in a test tracking sequence. The user then interactively clicks four points of a polygon delineating the object to track. After doing so, the "Track" menu option tracks the object through the sequence while displaying the updated bounding box at each frame. Other menu options include "Stop" for pausing the tracker, "Step>" for single stepping the tracker forward by a single frame, and "Rewind" to bring up the first frame in the sequence again for another tracking run with the same initial polygon. If "Log" is selected, results are logged for each frame for later replay. "Predict" turns on motion prediction to enable tracking through occlusion.

Functionality of the testbed includes:

**Output Logging:** When logging is turned on, tracking results for each frame are stored in ascii log files for later review. Included in each log file is the current bounding box of the object in that frame and a binary bitmap that specifies which pixels belong to the object versus the background. The log file format is compatible with the automated scoring mechanism on the tracking evaluation web site (Section 3).

**Replay of Experiments:** Through use of the log files, previous tracking sessions can be read back into the testbed and replayed for review. Furthermore, the replay can be stopped at any frame and tracking restarted from that state using a different algorithm or set of parameters.

**Batch Mode:** Although typical usage will be interactive, a batch tracking mode is invoked when the user opens a file that specifies a list of multiple sequences and polygons for tracker initialization. When batch mode is invoked, the testbed performs tracking on each sequence in the list, without interruption, until the end of the list is reached. This provides a way to run a set of tracking experiments overnight without intervention.


*Figure 1: Screen capture of the real-time C tracking testbed, showing the tracked object bounding box and menu of interactive options displayed across the top of the graphical user interface.*

**Baseline Algorithms:** A set of baseline tracking algorithms is provided with the testbed. Many algorithms are available (see Table 1). For example, "Template Match" refers to normalized correlation template matching. "Basic Mean Shift" is the algorithm from [5]. "Variance Ratio" refers to an algorithm from [4] that selects a color tracking feature that maximizes the separability between feature histograms computed from the foreground target and its surrounding background region, prior to running mean-shift. The adaptive version of this algorithm reruns the feature selection process at every $10^{th}$ frame, otherwise the process is performed only once (nonadaptive). Our main effort at this time is adding new baseline algorithms to the list.

*Table 1: Sample timings for baseline algorithms in the tracking testbed, in frames per second. Each timing is shown both with and without the KLT-based motion prediction module turned on (see Section 2.2). Timings are from an Intel Pentium-4, 2.4GHz machine with 1 GB of RAM.*

| Tracker | No Prediction (KLT off) | Prediction (KLT on) |
|---|---|---|
| Fg/Bg Histogram Shift | 21 | 7 |
| Basic Mean shift | 18 | 7 |
| Template Match | 20 | 7 |
| Variance Ratio | 20 | 7 |
| Variance Ratio Adaptive (Every 10 frames) | 17 | 7 |
| Peak Difference | 20 | 6 |
| Peak Difference Adaptive (Every 10 frames) | 15 | 5 |

**Algorithm API:** As a side benefit of integrating multiple tracking algorithms we have developed a modular API that allows researchers to integrate their own tracking algorithms into the testbed. Three basic functions need to be provided to integrate a new algorithm into the testbed:

TrackerInit (image, box, mask) –invoked to initialize a new object track. Arguments are the initial image frame, initial bounding box, and initial object bitmap.
TrackerNextFrame (image, &box, &TrackResult) – track target into the next frame, and return the tracking result. Input arguments are the next image frame and a predicted bounding box of where the object will be (see the motion prediction discussion to follow). Outputs are the new bounding box and a structure containing object bitmap, tracking confidence score, and an occlusion detection flag (for algorithms that can compute one).
TrackerCleanup() –End of experiment. Release allocated memory.

We encourage researchers to make their code available to others by integrating it into the testbed for release in later versions.

### 2.2 Motion Prediction Module

Two common cases of failure in airborne target tracking are: 1) sudden, large camera motions (for example, a panning or zooming motion); and 2) occlusion of the tracked object behind buildings or foliage. The tracking testbed includes a motion estimation and prediction module that addresses both of these issues by estimating and compensating for apparent motion of both the scene background and tracked foreground object. Parametric estimates of background scene displacements provide coarse predictions of where the object should be in the current frame based purely on the effects of camera motion. Foreground motion modeling fine tunes these estimates by adding a further displacement due to predicted object motion. Background motion estimation and foreground object motion prediction are coupled. In fact, we hypothesize that a constant velocity model is adequate for modeling object motion once the background camera motion has been compensated for. This hypothesis is validated later in this section.

The motion prediction module allows tracking through temporary, total occlusions. An occlusion event is flagged when tracking confidence falls below some percentage of the expected tracking confidence score. During occlusion, a model of the object motion is extrapolated forward in time until the object reappears and is reacquired (Figure 2).


*Figure 2. The testbed contains a motion prediction module that helps track objects through occlusion. The prediction method assumes object motion has constant velocity after compensating for affine motion of the scene background.*

We estimate background camera motion by fitting a global parametric motion model to sparse optic flow. The Kanade-Lucas-Tomasi (KLT) feature tracker [2] is used to match corner features between adjacent pairs of video frames to obtain a sparse estimate of the optic flow field. For each corner feature, the method solves for a subpixel translational displacement vector that minimizes the sum of squared intensity differences between an image patch centered at the corner and a patch in the subsequent frame centered at the estimated translated position.

A global, six parameter affine motion model is fit to the observed displacement vectors to approximate the flow field induced by camera motion and a rigid ground plane. Higher order motion models such as planar projective could be used, however the affine model has been adequate in our experiments due to the large sensor standoff distance, narrow field of view, and nearly planar ground structure in these aerial sequences. We use a Random Sample Consensus (RANSAC) procedure [7] to robustly estimate affine parameters from the observed displacement vectors. The method repeatedly selects a random set of three point correspondences, solves for the affine transformation induced by them, and counts the number of other correspondences that can be explained by the resulting transformation. The largest such set of correspondences is chosen to represent the statistical "inliers", and a least-squares solution applied to these inliers forms the final global affine estimate. The benefit of using the robust RANSAC procedure is that the final least squares estimate is not contaminated by incorrect displacement vectors, points on moving vehicles, or scene points with large parallax.

It is important to note that we do not do explicit image warping to produce stabilized image sequences. Instead, we compensate for camera motion by predicting apparent optic flow within an image region local to the object, so that it can be added to the vector that predicts where the object will go based on the constant velocity assumption. This simple linear algebra computation is far less expensive than image stabilization via warping.

It is also important to note that the motion prediction mechanism presented here is an optional feature of the tracking testbed. The user can turn off motion prediction completely, or can substitute another approach by implementing it within their own algorithm.

**Validating Affine-Compensated Constant Velocity**

The motion prediction module assumes the object travels with constant velocity after first compensating for affine background motion. To validate this assumption, we tested a simplified three-frame computation method (the actual motion prediction module in the testbed uses a sliding window of N previous object centroids to more robustly estimate current velocity).

We model the observed displacement of a target from one video frame to the next using two terms (see Figure 3). The first term, d_camera, models the displacement that would have been observed if the object had been stationary, or in other words, only due to motion of the camera. Camera motion includes proper movement such as forward translation of the airplane or rotation of the camera ball, and apparent motion due to changes in focal length during a zoom operation.

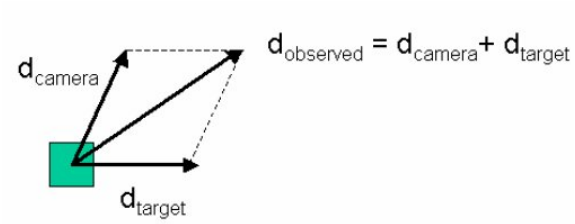


*Figure 3: Decomposition of observed displacement of a target between two video frames into terms based only on motion of the camera and motion of the target.*

The second term, d_target, models the displacement that would have been observed if the camera had been stationary, and thus assumes the displacement was due solely to motion of the object. To estimate the object-only displacement term we assume a constant velocity model, meaning the vehicle is traveling in a "steady-state". Of course, in reality the vehicle may be speeding up or slowing down, but as long as the change in acceleration is slow a "constant velocity + noise" model should be sufficient.

Figure 4 illustrates the terms involved in predicting where an object should be in the current frame based on a constant velocity model, given that we must first adjust for camera motion before we can compute the vehicle's velocity. The formula to predict the location $P_t$ of the object in the current frame *t*, given its previously observed positions $P_{t-1}$ and $P_{t-2}$ in the last two frames, is

$$P_t = T_t^{t-1} * \left[P_{t-1} + \left(P_{t-1} - \left(T_{t-1}^{t-2} * P_{t-2}\right)\right)\right]$$

where $T_{t-1}^{t-2}$ is the affine motion between frames *t*-2 to *t*-1, and $T_t^{t-1}$ is the affine motion between frames t-1 and t.

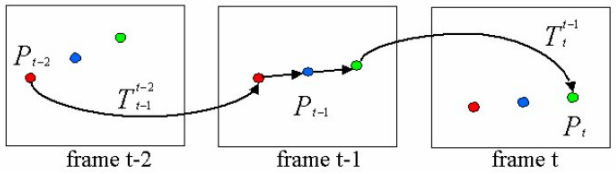


*Figure 4: Prediction of current location of target, Pt, based on its previous two observed positions, and computed inter-frame displacements due to camera motion.*

Once the object velocity is computed between two frames, prediction of object location based on constant velocity can be extrapolated into subsequent frames without further observation of the object position, as long as camera motion is estimated between each pair of frames. This multi-frame "blind" prediction of target location is important, since it forms the basis of tracking through occlusion.

We have used a large set of hand-labeled aerial tracking sequences to validate this motion prediction method. In these sequences, objects in every 10th frame have been outlined by polygons. The sequences include a variety of object resolutions (lens zoom factors), camera motions, and relative object motions. We test the predicted location of an object polygon centroid in frame t given its known location in frames t-2 and t-1 and estimates of inter-frame affine camera motions computed from sparse optic flow. [Since ground truth is only available for every 10th frame in the sequences, frame t-1 is actually 10 frames in the past, and t-2 is 20 frames in the past. Thus, the test is more demanding than the frame-to-frame prediction that occurs in the on-line tracking system.]

Figure 5 plots two-frame displacements of 13,852 target centroids with no motion compensation (left plot), after

compensation for affine camera motion only (middle), and after compensation for both camera motion and constant velocity object motion components (right). Since displacement scales with respect to zoom factor, we use a simple method for normalizing across zoom (vehicle size) by measuring centroid displacement in pixels divided by the square root of the number of pixels in the object image polygon. When an object is relatively compact, as is the case with vehicles, displacements within its borders map roughly to a circle of radius 1, which is overlaid on the plots. For tracking methods based on gradient descent (e.g. Lucas-Kanade method and Mean-shift), we want the observed displacements to fall within this circle so that the predicted object polygon in the next frame overlaps well with the true object polygon. Points outside of radius 1 represent cases where a gradient descent tracker may fail, and points outside of radius 2 represent cases that are almost guaranteed to fail, since the predicted object polygon does not overlap the new object polygon at all.

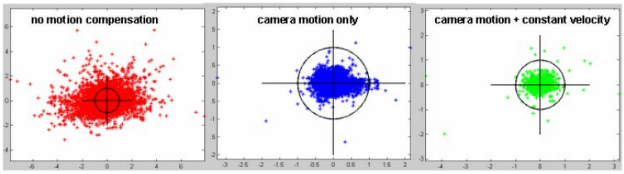


*Figure 5: Validation of camera motion compensation and constant velocity location prediction. Units in the plots are centroid displacement / sqrt(pixels on target). Each plot is overlaid with a circle of unit radius. These plots result from 13,852 motion prediction tests.*

As we see from Figure 5, without motion compensation there are many cases of interframe displacements that are large enough to cause tracking failure. In contrast, compensating for affine camera motion brings the vast majority of the location predictions within the circle of radius 1, thereby bounding the displacement to a degree where success is likely. We see in the far right plot the effects of applying both camera motion AND constant velocity target motion prediction. Although a few more points fall outside the circle of radius 1, the clustering of compensated points around the origin is actually tighter (major axis standard deviation of 0.13 as opposed to 0.20 using only camera motion compensation, and 0.83 with no compensation at all). The results clearly validate that affine background motion compensation followed by constant velocity motion prediction is an adequate motion prediction model for airborne tracking, and that it can greatly increase the likelihood of success of simple gradient descent target search methods.

## 3. Evaluation Web Page Development

We have developed an on-line, tracking evaluation web site to facilitate third-party self-evaluation of tracking systems on airborne video data. The inspiration for the site is the successful Middlebury stereo evaluation web page [8]. Our evaluation site provides: 1) ground-truth datasets for tracking experiments; 2) the tracking testbed software; 3) a mechanism for uploading and automatically scoring tracking results; and 4) a table showing the user's score ranked in relation to other algorithm submissions. See Figure 6.

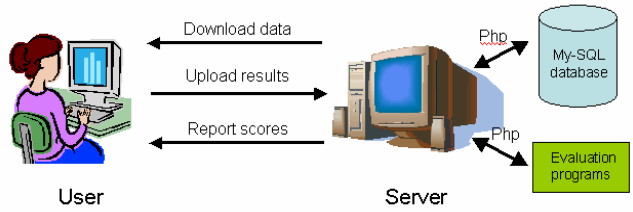


*Figure 6. Framework for development of a tracking evaluation web page. PHP-enabled web scripts interface with a My-SQL database on a dedicated server, yielding great flexibility in designing and implementing web page functionality.*

The basic website framework is built on PHP web scripts on a dedicated Apache server. PHP is an open source, server-side, HTML-embedded scripting language used to create dynamic Web pages. PHP can perform callouts to C programs, giving us great flexibility in designing the web page functionality. However, the primary strength of PHP lies in its compatibility with many types of databases. In our evaluation setup, PHP interfaces with a My-SQL database. We determined that it would be necessary to have an underlying database to manage the large volume of test datasets and uploaded results submitted by the web page users. Furthermore, the database enables dynamic manipulation and tabulation of evaluation scores.

The evaluation website is account-based. Each time a user submits results for a different algorithm, they set up an account that maintains its own set of evaluation scores on the web server's MySQL database. To improve security for the website, we prevent automated account registrations by using visual captcha's – the user must read a distorted picture representing a string of letters and numerals that must be entered for verification. We also limit the size of file uploads to prevent malicious usage.

### 3.1 Evaluation Datasets

The aerial tracking datasets are a subset of public release data collected under the DARPA VIVID program. In selecting evaluation video clips, the goal has been to offer

a representative sample of object resolution, contrast, pose, and degree of occlusion, in both visible and thermal IR imagery. A sample sequence thumbnail page is shown in Figure 7.

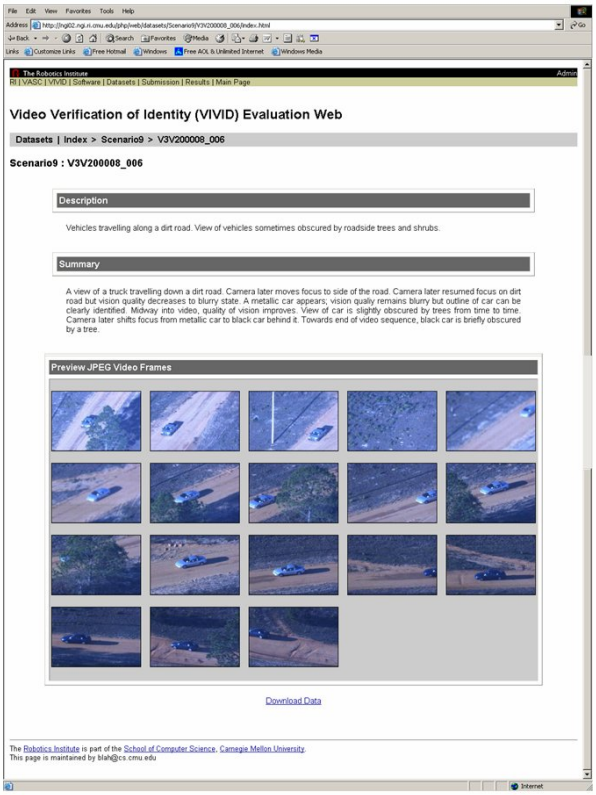


*Figure 7. A sample dataset thumbnail page. From this page, the user can download the dataset, which consists of a zipped sequence of jpeg image frames.*

The original video clips are avi movie files encoded via motion-jpeg. To remove any potential variability due to use of different decoders, we distribute the video frames as sequences of numbered jpeg image files. We use a program called Videomach to "explode" avi files into color image frames, postprocess the decoded frames with a de-interlace filter that replaces every odd scan line with new values bilinearly interpolated from neighboring even scan lines, and store the results as a numbered sequence of images in jpeg file format. The de-interlace filter removes even-odd scan-line artifacts due to sensor interlace, allowing more accurate delineation of the boundaries and bounding boxes of fast moving objects.

### 3.2 Baseline Algorithms

Distribution of a set of baseline algorithms into the public domain is accomplished through the open source tracking testbed described in the previous section and available for download from the web page. Making baseline code available is valuable because it encourages and facilitates scientific repeatability. Additionally, it reduces the cost of entry into the evaluation "competition" since people can start with one of the baseline algorithms and tweak its modules to improve performance.

### 3.3 Evaluation Metrics

The automated scoring algorithm is written in C++. It uses ground truth files as a benchmark against which to compare user-generated tracking results. Two types of metrics are used: tracking success rate and shape accuracy of computed foreground masks (which can be reported just as a bounding box by algorithms that do not attempt to do shape segmentation). Specifically, the evaluation algorithm uses five criteria to rate and rank submitted results:

1) percentage of dataset tracked: this is the number of frames in which the user tracked the object before losing it, divided by the total number of frames. The track is considered to be lost if the bounding box does not overlap the ground truth bounding box at all. The first such occurrence terminates the evaluation. This implies that we do not allow the user's tracker to reacquire the target after tracking failure.

2) average overlap between bounding boxes: this is the percentage of overlap between the user's bounding box and the bounding box identified by ground truth files. This is only computed over the portion of the dataset reported in the percentage score above.

3) average overlap between bitmaps within overlapping bounding box area. This is computed in the area of intersection between the user bounding box and the ground-truth bounding box. This criterion measures the accuracy/similarity of the bitmap (binary foreground mask) specified by the user's algorithm to the ground truth object bitmap.

4) average distance transform focused on ground-truth object. As motivated in the next section, we choose the chamfer distance as a measure of distance between two binary bitmaps. This version of the score uses the ground truth object bitmap to compute the distance transform against which the user object bitmap is scored.

5) average distance transform focused on user-identified object. Same as above, but the user-supplied bitmap is now used to compute the distance transform against which the ground truth bitmap is scored. We need both versions since the chamfer distance is not symmetric. Alternatively, we could compute the average of the two directed chamfer distances to generate a symmetric error.

After scoring, a table showing the user's ranking with respect to other submitted algorithms is automatically generated and displayed (Figure 8).

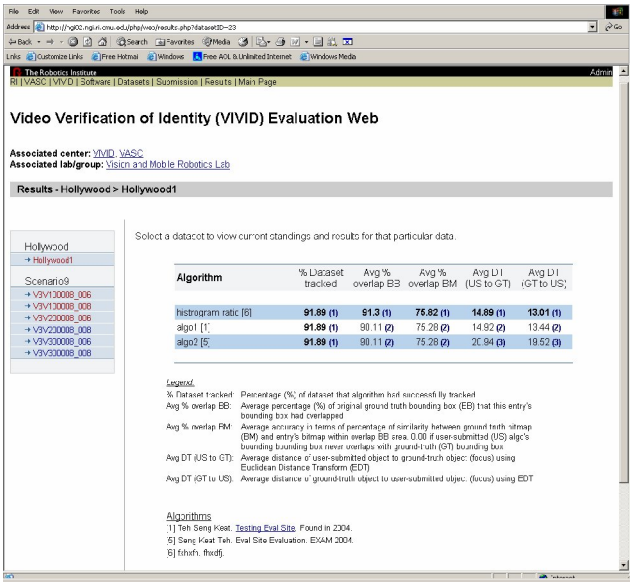


*Figure 8. Upon uploading a set of results to the web site, an automated scoring mechanism is invoked that compares the user's results to a ground truth dataset for that sequence. The algorithm score is displayed in a table in relation to the results achieved by other submitted algorithms.*

### 3.4 More on Metrics – Position and Shape

We want to have a metric describing how well the tracker has found the tracked object in the current frame. If we represent the object as a point (centroid) location, a natural measure for goodness of fit is distance to the ground truth location. However, a tracking algorithm should know not just the object location but also the spatial extent of the object in the image. It is common to display the extent of an object hypothesis with a bounding box overlaid on the image. It is therefore natural to consider percentage of overlap of bounding boxes as a joint measure on similarity of location and extent. Bounding boxes are also fairly easy to ground truth in the image. However, Figure 9 shows that overlap of bounding boxes can be a poor description of similarity for objects that are not oriented along scan lines of the image. For the pathological example shown on the left of the figure, two hypotheses with very different shapes have identical bounding boxes and, incidentally, identical centroid locations as well.

We propose to represent extent and shape of an object by a bounding box and a binary bitmap, and to measure accuracy of an object hypothesis by its similarity to a ground truth object bitmap. Several measures have been devised in the past to compute similarity of segmented shapes. If a bitmap mask is treated as a binary classification of pixels into foreground and background classes, one similarity score between hypothesis and ground truth masks is TP/(TP+FP+FN) where TP is the number of correctly labeled foreground pixels, FP is the number of pixels incorrectly labeled as foreground, and FN is the number of pixels incorrectly labeled as background. This score ranges from 0 when there is no overlap of hypothesis and ground truth, to 1 when they are exactly the same. The trouble with this score is the inability to make fine distinctions between bitmaps that don't overlap: a hypothesis bitmap that does not overlap the ground truth but is nevertheless nearby should not be penalized as much as a bitmap that is located far away (Figure 9). [Since we terminate a tracking experiment when bounding boxes no longer overlap, ability to measure this distance is moot. However, the drawback described is still valid.]

The evaluation web page scoring mechanism uses the *chamfer distance* to jointly measure similarity of bitmap positions and shapes. Chamfer distance is the average of the minimum distances from foreground pixels in one bitmap to foreground pixels in another. Chamfer distance can be computed efficiently based on algorithms for computing distance transforms. Note that chamfer distance behaves sanely on the two pathological cases we have shown in Figure 9. The chamfer distance for two objects that completely overlap is 0. The chamfer distance for two objects that do not overlap at all is proportional to the distance between them. In cases of partial overlap the score quantifies the trade-off between position and shape accuracy.

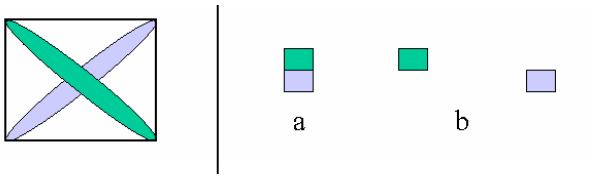


*Figure9. Pathologies of some simple hypothesis similarity measures. Left) the green object is not very similar to the blue object, but their centroids coincide and their bounding boxes overlap perfectly. Right) cases a and b show two hypotheses that do not overlap, and thus would both score 0 using some similarity measures. We claim situation a should have a higher score than situation b. The proposed chamfer similarity measure behaves correctly in each of these examples*

### 3.5 Generating Hand-labeled Data

Any measure based on similarity of object bitmaps means that ground truth segmentation of the foreground object must be achieved. This requires significantly more human effort than just selecting bounding boxes. We have developed a simple Matlab ground truthing tool that

allows the user to interactively draw a polygonal outline around the object, using the mouse. The process is illustrated in Figure 10. When a frame is displayed, the user initially selects a region of interest containing the object. A zoomed in version is then displayed and the user proceeds to draw a polygonal boundary around the object. A simple sequence of left and right mouse clicks allows the user to add or remove points (based on the Matlab command "roipoly"), until they are satisfied with the object contour. The next frame to be labeled is then displayed.

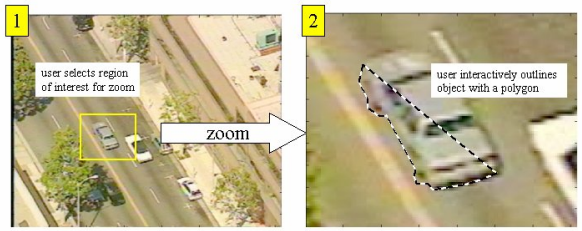


*Figure 10. Overview of our current process for hand-labeling ground truth object masks for tracking evaluation: 1) user selects a region of interest that contains the object, 2) a zoomed in image is displayed, and the user draws a polygonal boundary around the object using a sequence of mouse clicks. The resulting contour is then converted into a bounding box and a binary mask denoting the shape and location of the object.*

Figure 11 shows some samples of ground truth frames. The interactive outlining tool allows the user to quickly label a variety of cases, including unoccluded object boundaries, partial occlusions, complete occlusions (which is easy), and even cases where the object is split into multiple pieces by thin occluding objects. We are currently able to ground truth roughly 5 frames per minute, and currently we ground truth every 10$^{th}$ frame in a sequence.

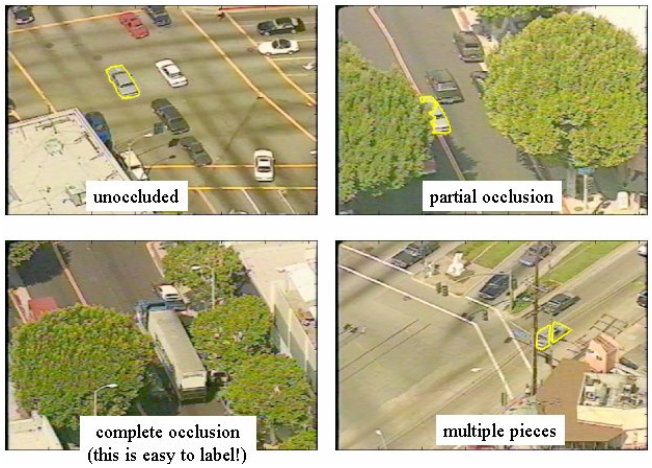


*Figure 11. Some sample ground truth labelings, ranging from unoccluded contours, partial occlusion, complete occlusion, and breaking into multiple pieces due to thin occluding objects.*

## 4. Conclusion

We have developed an open source tracking testbed and evaluation web site to encourage third-party experimentation and evaluation of tracking algorithms. By making these resources available we seek to make it easier for the tracking research community to compare performance of different algorithms on the same data. We also hope that researchers will contribute new algorithms to the growing library of baseline algorithms distributed within the testbed system.

Although the tracking testbed can be used on any video sequences, the current datasets on the evaluation page are focused on ground vehicle tracking from airborne video Our development effort was sponsored by a project for which these are the relevant datasets to consider. These sequences are also interesting in their own right due to the challenges of a constantly moving background. In the future we may add datasets of other types, such as static camera surveillance scenarios, and would welcome being a mirror site for the PETS benchmark datasets.

## Acknowledgements

This work was funded by the DARPA/IXO VIVID project under contract NBCH1030013.

## References

1. Beveridge, R., *Evaluation of face recognition algorithms web site*. URL: http://cs.colostate.edu/evalfacerec
2. Birchfield, S., *KLT: an Implementation of the Kanade-Lucas-Tomasi Feature Tracker*, 1997, URL: http://www.ces.clemson.edu/~stb/klt/
3. Bradski, G. "OpenCV: Examples of Use and New Applications in Stereo, Recognition and Tracking", *The 15th Intl Conference on Vision Interface*, 2002. (URL: http://www.intel.com/research/mrl/research/opencv/)
4. Collins, R. and Liu, Y. "On-Line Selection of Discriminative Tracking Features, *IEEE Int'l Conference on Computer Vision*, Nice, France, Oct 2004, pp. 346-352.
5. Comaniciu, D., Ramesh, V. and Meer,P., "Kernel-Based Object Tracking," *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 25, No. 4, 2003
6. Ferryman, J., *IEEE Int'l Workshops on Performance Evaluation of Tracking and Surveillance*, 2000-2004. (URL: http://www.cvg.cs.rdg.ac.uk/VSPETS/)
7. Fischler, M. and Bolles, R., "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Comm. of the ACM*, Vol 24, 1981, pp. 381-395.
8. Scharstein, D., *Middlebury stereo vision page*, URL: http://www.middlebury.edu/stereo/