Persistent Tracking for Wide Area Aerial Surveillance

Jan Prokaj
J Gradient
jan@jgradient.com

Gérard Medioni
University of Southern California
medioni@usc.edu

Abstract

Persistent surveillance of large geographic areas from unmanned aerial vehicles allows us to learn much about the daily activities in the region of interest. Nearly all of the approaches addressing tracking in this imagery are detection-based and rely on background subtraction or frame differencing to provide detections. This, however, makes it difficult to track targets once they slow down or stop, which is not acceptable for persistent tracking, our goal.

We present a multiple target tracking approach that does not exclusively rely on background subtraction and is better able to track targets through stops. It accomplishes this by effectively running two trackers in parallel: one based on detections from background subtraction providing target initialization and reacquisition, and one based on a target state regressor providing frame to frame tracking. We evaluated the proposed approach on a long sequence from a wide area aerial imagery dataset, and the results show improved object detection rates and ID-switch rates with limited increases in false alarms compared to the competition.

1. Introduction

Persistent surveillance of large geographic areas from unmanned aerial vehicles (drones) allows us to learn much about the daily activities in the region of interest. This is not only useful for security applications, but it also has the potential to enable real-time traffic optimizations and map updates. Sensors that capture imagery for persistent surveillance are multi-camera, large format (60-100 megapixels), have low sampling rate, provide limited resolution on targets, and are in grayscale (see Figure 1 for an example). Therefore, understanding people's activities and movements requires multiple target tracking algorithms that can cope with these characteristics.

Nearly all of the approaches addressing tracking in this imagery, called wide area aerial imagery, are detection-based and rely on background subtraction or frame differencing to provide detections [14, 19, 17, 11]. Recent work shows detection based tracking approaches are powerful

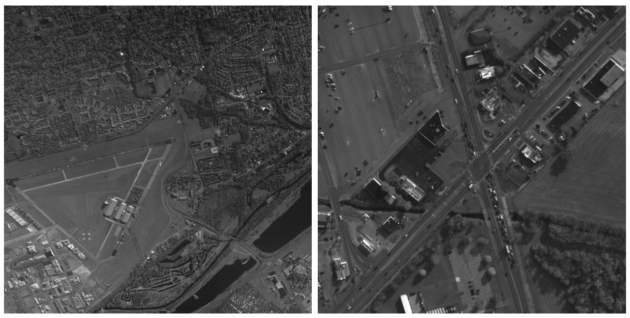


Figure 1. Wide area imagery, full frame (left) and detail (right).

[15, 4], but they assume a target detector with reasonable performance can be learned. In wide area imagery, where the resolution of each target is limited (about 20×10 pixels), training such a detector is difficult. Therefore, background subtraction or frame differencing is used instead. This, however, makes it impossible to track targets once they slow down or stop, because background models are often built over short time scale to avoid introducing errors from parallax, lighting changes, or inaccurate stabilization (drift). Our goal is to achieve *persistent* tracking, and losing targets every time they stop is not acceptable.

In this work, we present a multiple target tracking approach that does not exclusively rely on background subtraction and is better able to track targets through stops. It accomplishes this by effectively running two trackers in parallel: one based on detections from background subtraction providing target initialization and reacquisition, and one based on a target state regressor providing frame to frame tracking. The detection based tracker provides accurate initialization by inferring tracklets over a short time period (5 frames). The initialization period is then used to learn a non-parametric regressor based on target appearance templates, which is able to directly infer the true target state from a given target state sample in every frame. When the regressor based tracker fails (loses a target), it falls back to the detection based tracker for reinitialization.

Our primary contribution in this work is a multiple target tracking approach for wide area aerial surveillance imagery that is better able to track targets through stops and brings us closer to persistent tracking. We evaluated the proposed approach on a real sequence from wide area aerial surveillance imagery and the results show an increased object detection rate, decreased ID-switch rate, and decreased track fragmentation, compared to competing approaches.

2. Related Work

One of the earliest works on tracking targets in wide area imagery is by Perera *et al*. [14]. Detections are obtained using background subtraction, formed into short tracklets using nearest neighbor association, and the tracklets are linked using the Hungarian algorithm. The noisy nature of background subtraction is recognized as a problem, generating split and merged detections of targets. These are handled by generating a set of data association hypotheses and approximately solving for the best hypothesis by iteratively augmenting the correspondence cost matrix. This approach may not be scalable, and its reliance on background subtraction for detection would make it difficult to handle stopping targets.

In the approach of Xiao *et al*. [23], there is some attempt to track stopping targets. In addition to the background difference, a template-based appearance model and shape model are used to generate three candidate detections for every target. Detections are then associated with tracks using the Hungarian algorithm. Improved association is obtained by considering road and spatial constraints. However, the spatial constraints require the enumeration of all track-detection pairs, which is costly (in addition to the high complexity of the Hungarian algorithm), and the road constraint, while useful, requires the prior knowledge of a road network.

Reilly *et al*. [19] also use the Hungarian algorithm for association of detections, but propose to increase its efficiency by dividing the image into cells and computing the associations within each cell. The matching cost between targets takes into account spatial proximity, velocity orientation, orientation of the road, and local context between cars. The road constraint and local context constraint, while useful in dense traffic, would be less reliable or difficult to estimate in sparse traffic scenarios. Stationary targets are not considered, and in fact would be difficult to associate with the proposed velocity orientation constraints.

In a more recent work of Keck *et al*. [11], classic multiple hypothesis tracking on detections obtained with three-frame differencing is adopted, and implemented in a real-time, distributed, architecture. This presumably associates detections better than the frame-to-frame Hungarian algorithm, but there is no provision for stationary targets.

In [17], Prokaj *et al*. presented another approach that uses more than one frame of data to infer tracks. This approach uses long temporal windows of 8 seconds for inference, but is able to significantly reduce the space of possible data associations by efficiently pruning detections that are inconsistent. Detections from background subtraction are used there as well.

Most of the detection based approaches considered so far would be able to track stationary targets, if detections for those targets were obtained using an appearance-based classifier rather than background subtraction or frame differencing. Even though we believe that this is difficult and unreliable in our domain, there have been some attempts to do this [6, 12]. Doretto and Yao [6] obtained very good vehicle detection results in aerial imagery, even when the vehicles were small. However, their experiments showed that the most important features were color, which is not available in our imagery. In [12], an SVM with multiple kernels based on HoG and Haar features was trained, and good vehicle classification results were obtained. At the same time, in a follow up work, some of the same authors [20] argue the need for road network context in order to reliably detect vehicles. Therefore, we conclude that a reliable appearance-based detection of targets in our domain remains out of reach.

Alternative formulations of the multiple target tracking problem using network flows/Linear programming have shown excellent results in pedestrian tracking [24, 15, 4]. However, these methods have a few inherent assumptions unsuitable in our domain. They have weak motion models, often require a prior specification of locations where objects enter the scene and exit, and they usually express the association cost in terms of overlap of two detections, which will not work in our domain where the frame rate is low.

The benefits of using a regressor rather than a classifier in tracking have been shown by Williams *et al*. [22]. A regression model allows the direct inference of state from appearance features, which makes it very efficient, because there is no need to exhaustively perform classification around the predicted position of the target. Furthermore, it is more accurate and less susceptible to drift, because it actually learns the necessary adjustments to correct a displaced (or drifted) target state from appearance features. Recently, this "structured output" was shown to outperform competing approaches [7].

3. Approach

Our goal is to achieve persistent tracking, which means being able to track targets as soon as they begin to move, and as long as they are visible. This rules out exclusively relying on background subtraction or frame differencing, because these operators would not detect targets when they become stationary. Target appearance modelling is needed. Since the resolution of targets in our imagery is limited, learning a target *class* appearance model is difficult. Therefore, we turn to learning a target *instance* appearance model. We discuss our model in Section 3.1.2.

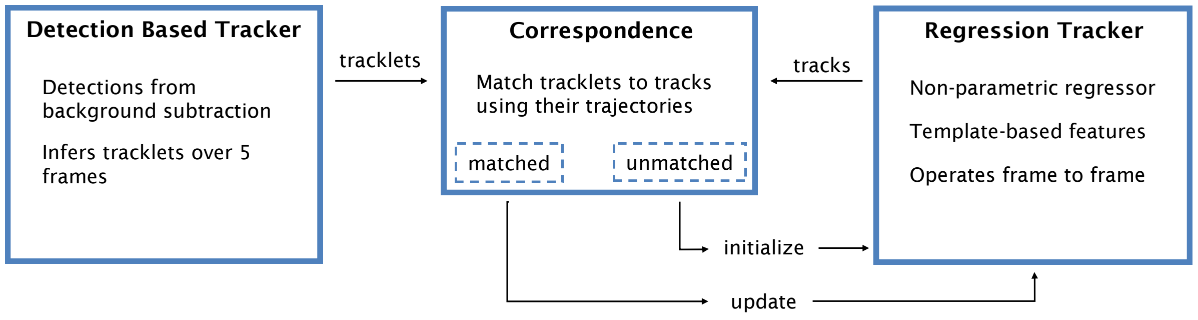


Figure 2. Overview of our approach. There are two trackers running in parallel, complementing each other.

Whatever appearance model we choose, we need to be able to initialize it and update it when target's appearance changes. This creates a chicken-and-egg problem, because we are trying to learn an appearance model in order to track a target, but in order to learn this model, we need to be able to detect the target in the first place. Similarly, when target's appearance changes, we would like to update the model with the new appearance, but we can not detect the target with the new appearance, until we have updated the model. To solve both of these initialization and update problems, we propose to use an additional tracker based on detections from background subtraction, which has minimal appearance modelling. This tracker runs in parallel to and complements the main tracker based on appearance modelling. We call the main tracker based on appearance modelling "regression tracker." The detection based tracker [17] operates on a sliding window of 5 frames, and generates tracklets that correspond to moving targets every frame. These tracklets are used to initialize and update the regression tracker.

Our approach is illustrated in Figure 2. The key components are a detection based tracker, which is explained in [16, 17], a regression based tracker, explained in Section 3.1, and tracklet-track matching and failure detection, explained in Section 3.2. We now explain each of these components in detail.

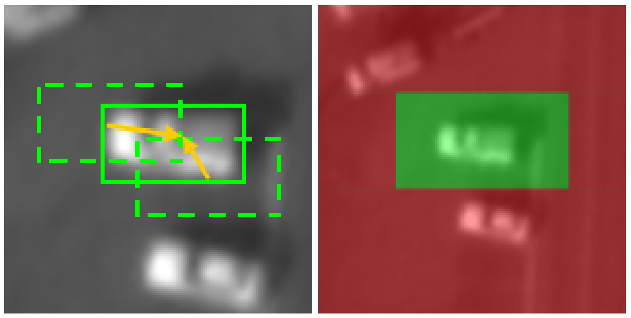
3.1. Regression Tracker

We are given a moving target's location in the first 5 frames and our task is to track the target as long as possible, even through stops. The usual way to proceed is to learn a classifier (appearance model) from the initial examples, predict (sample) the target's state in the next frame using a motion model, apply the classifier on each sample, update the target's state with the sample maximizing appearance model likelihood or posterior, and update the classifier (and motion model if applicable) [2, 3, 10]. We deviate from this framework slightly to achieve better performance.

First, unlike in other domains, motion modelling is important here, because we cannot scan the entire frame (there are hundreds of targets, each with limited resolution) and we cannot assume the target is going to be in approximately the same location (the frame rate is low). Fortunately, our targets are vehicles, which have a relatively predictable motion (see Sec. 3.1.3). Nevertheless, there will be instances where the prediction from the motion model is not going to be accurate. In those cases, we need to rely on the appearance model to find the true target state from the noisy set of samples.

Second, similarly to [7], we argue that a binary classifier is not the appropriate way to model small transformations (translations, rotations) of the target. We believe that a much better way to proceed is to actually learn the effect of the displacement of target's state on appearance. This is possible when training a regressor rather than a classifier, and was first introduced by Williams *et al.* [22]. During regressor training, we provide examples labeled with displacements of the target state, such as $(\Delta x, \Delta y, \Delta\theta)$ instead of class labels $(0/1)$. During testing, when we are given a predicted state of the target, we can use the appearance of the prediction to directly estimate the correct target state (using the displacement returned by the regressor). For example, if we give a regressor examples labeled with translations in the range $[-4, 4] \times [-4, 4]$, and then during testing we happen to sample (-3,2) pixels away from the true target location, our regressor will return (-3,2). This is in contrast to a yes/no answer output by a classifier, which does not tell us as much. This is illustrated in Figure 3a.

The primary advantage of a regressor, as first noted by [22], is efficiency. There is no need to take many samples or do exhaustive search, because every sample gives us information about the target's state. In the ideal case, only one sample is necessary to determine where the target really is. However, the target must be at least partially visible in the location we want to regress. If the target is not visible, in general there is no information one can use to determine the target's location, and the regressor's output would be meaningless. In [22], this problem is solved by training a classifier in addition to a regressor to determine when the target is completely out of bounds. In this work, we use a more robust unsupervised approach to solve this problem. Computational savings are still made, but even without the savings, we believe that for the case of the target being partially



(a) (b)

Figure 3. A regressor is able to output the displacement to the true target's state, whereas a classifier is only able to say yes/no (left). However, this only works for samples close to the target (right).

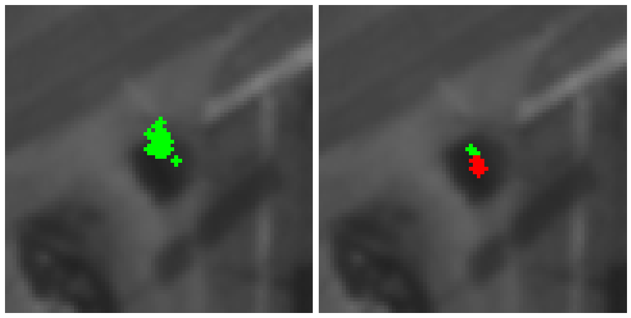


Figure 4. A regressor improves target state estimates. The left image shows samples before a regressor is applied. The right image shows the same samples after a regressor is applied. Crosses colored in red denote the final set of valid samples.

visible, a regressor is an inherently more powerful function than a classifier (much less likely to cause drift). This is illustrated in Figures 3 and 4.

The disadvantage of using a supervised approach to classify samples as being valid or invalid for regression is that we need to be careful about what training examples we use in training. The size of the training set is limited by computational constraints, which makes it especially difficult to choose good negative examples, since their variation is large. Therefore, we use the following unsupervised approach, which we have informally observed to perform better than using an SVM as in [22].

The key idea is to recognize that the output of the regressor is going to be more trustworthy when multiple samples taken around the search region generate the same target state. When the samples are taken from areas where the target is not visible, the output of the regressor is going to be unpredictable and not going to create clusters. Accordingly, we would like to find the largest cluster of target state estimates with the smallest variance. In practice, we build a histogram of the target state estimates, where the bins are of some small fixed size. We then choose the bin with the highest number of samples as the best cluster. If the number of samples there is higher than some threshold, these samples are then all considered valid.

Figure 4 illustrates this approach. The left side of the figure shows that the motion model assumes the target is moving faster than it really is (the vehicle is actually coming to a stop). As a result, the samples are biased towards the front of the vehicle and many are not even on the target. After applying a regressor, the variance of the samples is significantly decreased. Furthermore, the valid samples identified using our unsupervised approach increase the precision even more, correctly locating the center of the target.

The regression tracker algorithm is summarized in Algorithm 1. The main components of the tracker are a regressor, feature extractor, and a motion model. We explain these components next.

Algorithm 1 Regression Tracker

```
Input: $\mathbf{z}_t$ = target state in frame $t$, $M$ = motion model,
       $\phi$ = feature extraction, $reg$ = trained regressor
Output: $\mathbf{z}_{t+1}$ = target state in frame $t+1$

    // sample from the motion model
 1: $S = \{\mathbf{x} \mid \mathbf{x} \sim M(\mathbf{z}_t)\}$

    // regress each sample
 2: $R = \{\mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma) \mid \mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma) = reg(\phi(\mathbf{s}))\ \forall \mathbf{s} \in S\}$

    // find the valid subset of samples
 3: $R_\mu = \{\boldsymbol{\mu} \mid \mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma) \in R\}$
 4: $h = \text{histogram}(R_\mu)$
 5: $b_{\max} = \arg\max h(b)$
 6: $V = \{\mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma) \mid \boldsymbol{\mu} \in h[b_{\max}] \wedge \mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma) \in R\}$

    // check the uncertainty of the subset
 7: $V_\mu = \{\boldsymbol{\mu} \mid \mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma) \in V\}$
 8: if $|V_\mu| < t_1 \vee \det(\text{cov}(V_\mu)) > t_2$ then
 9:     return

10: $p(\mathbf{x}) = \sum_{\mathcal{N}\in V} \mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma)$
11: $\mathbf{x}* = \arg\max p(\mathbf{x})$

    // reduce false alarms
12: check frame difference is consistent with est. motion

13: $\mathbf{z}_{t+1}$ = updateMotionModel($\mathbf{x}*$)
```

3.1.1 Non-Parametric Regression

In regression we are trying to estimate the quantitative value of a function f at a previously unseen point $\mathbf{x}_0$. In our application, we are trying to estimate the target's displacement given some features extracted at a sample image location. There are various forms of regression, each with different assumptions about the structure of the feature space, and number of parameters. An RVM was used by Williams *et al*. in [22]. Its advantage is that it is fully probabilistic, but it comes at a cost of more computationally expensive training. In this work we use a simple, non-parametric (kernel) regression called the Nadaraya-Watson kernel-weighted average [8]. The regressor is of the form

$$f(\mathbf{x}_0) = \frac{\sum_{i=1}^{N} k(\mathbf{x}_0, \mathbf{x}_i)\mathbf{y}_i}{\sum_{i=1}^{N} k(\mathbf{x}_0, \mathbf{x}_i)} \tag{1}$$

where $(\mathbf{x}_i, \mathbf{y}_i)$ is the training set, and k is the kernel function. In our application, $\mathbf{x}_i$ would be a feature vector and $\mathbf{y}_i$ would be the corresponding displacement in the target's state, $(\Delta x, \Delta y, \Delta\theta)$ when regressing target's translation and rotation. In addition to the average, we can estimate uncertainty in the regressor's ouput as the kernel-weighted covariance. Therefore, our regressor actually provides a Gaussian distribution $\mathcal{N}$ for each sample it evaluates.

There are several choices for the kernel function, and we adopt one based on the KL-divergence [13], since our features are effectively probability distributions (see below). The kernel function is

$$k(\mathbf{x}_0, \mathbf{x}_i) = \exp(-\lambda KL(\mathbf{x}_0, \mathbf{x}_i)) \tag{2}$$

where λ is a smoothing parameter determining the width of the local neighborhood, and KL denotes the Kullback-Leibler divergence. We determine the value of λ empirically.

3.1.2 Features for Regression

Some of the recent state of the art methods have relied on templates as their features of choice [10, 5]. We follow this work, and also use template-based features for regression. The template model is built from the 5 examples obtained using the detection-based tracker. It is rotationally variant, and we model target orientation as part of our state. When estimating the template, we rotate all the examples to a canonical orientation and scale them to a canonical size of 20×10. The template is estimated as the average canonical image. When evaluating a test sample, it is also first transformed to this canonical frame.

The small size of our targets can be a problem when using templates, because a relatively large portion of the template can be background. To avoid the influence of the background, we also estimate a target shape mask using thresholded background difference images of the 5 examples. When measuring the difference between a template and a test sample, the difference is only calculated for the pixels inside the shape, thus minimizing the influence of the background. Figure 5 illustrates the template learning process.

Given this template, we would generate a training set for regression by displacing the template with known varying

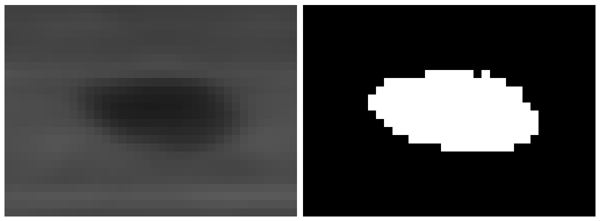


Figure 5. Target template as well as the shape mask is learned from 5 examples. The template is expressed in a canonical coordinate frame. Figure 4 shows the original target orientation and size.

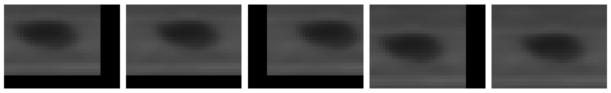


Figure 6. Subset of the displaced versions of the template used in regression. Each image has a corresponding shape mask.

amounts of translation and rotation. See Figure 6 for an illustration. During test time, we would measure the difference between a test canonical image and each of the training examples (displaced versions of the template), and measure the similarity of each training example to the test sample using a kernel function. This would work in theory, but since our targets are small with little texture, the regressor might have a difficulty recognizing the difference between small displacements. Therefore, we add more context. Instead of basing the kernel weight on the difference of a test sample with one training example, we base the kernel weight on the *distribution* of differences between several training examples.

The feature vector is computed as follows. Assume we are training a regressor to predict translations in the range $[-M, M] \times [-N, N]$. The number of displaced versions of the template and its corresponding mask is then $K = (2M + 1) * (2N + 1)$. Reasonable values of M and N are $M = 6$ and $N = 4$, giving $K = 117$. Let's denote each displaced template by $\mathbf{T}_i$ and each displaced mask by $\mathbf{S}_i$. Given a test sample $\mathbf{I}$ (transformed to the coordinate frame), the feature vector $\mathbf{x}$ is

$$\mathbf{x} = \left[g(\mathbf{I} - \mathbf{T}_0)^T g(\mathbf{I} - \mathbf{T}_0), \cdots, g(\mathbf{I} - \mathbf{T}_K)^T g(\mathbf{I} - \mathbf{T}_K)\right] \tag{3}$$

where

$$g(\mathbf{D}[j]) = \left\{ \begin{array}{ll} 10 & \mathbf{S}_i[j] \leq 128 \\ \min(|\mathbf{D}[j]|, 10) & \text{otherwise} \end{array} \right. . \tag{4}$$

The size of the feature vector is K, and each element is the sum of the squared differences between the test sample and a displaced template. However, the differences are truncated to be no larger than 10. Furthermore, for pixels outside of the shape mask, the difference is automatically 10. By normalizing this feature vector we get a probability distribution that could be used in our kernel function.

3.1.3 Motion Modelling

A motion model is used to predict the target's location in the next frame. Unlike previous approaches, which use the constant velocity motion model, we adopt a data driven approach. We use a tracking ground truth to generate examples of target's motion over a short time interval, and train a multi-variate regressor [21] on these examples for predicting the target's location. Specifically, we do the following.

For each ground truth track, we generate all possible fixed length sequences of target's position. Given a sequence of length T, we use the $T-1$ positions, expressed as 2D coordinates, as input (features) to the regressor and the last position as output. The target coordinates are normalized to achieve translation and rotation invariance. Translation invariance is achieved by subtracting the position of the target in the first frame of the sequence. Rotation invariance is achieved by rotating the resulting coordinates by the major direction of the target's motion. This direction is estimated from the displacement of the target between the first and the last input position. For example, for $T = 6$, the $2*(T-1)$ dimensional input vector $\mathbf{s}$, and 2 dimensional output vector $\mathbf{y}$ is then

$$\mathbf{s} = [0\,0\,x_2\,y_2\,x_3\,y_3\,x_4\,y_4\,x_5\,0] \tag{5}$$
$$\mathbf{y} = [x_6\,y_6]\,. \tag{6}$$

Since the first 2 elements, as well as the last element, of $\mathbf{s}$ are always 0, they do not need to be considered as features in training. We used a small ~ 700 element dataset for training the MVRVM in this paper, and obtained 19 "support vectors" after training. Note that when physical units are used (meters), this model only needs to be trained once, and can be theoretically applied in any dataset.

Sampling from this motion model is equivalent to sampling from a Gaussian distribution with a mean corresponding to the predicted position of the target in the next frame. The covariance of the Gaussian distribution should reflect the uncertainty in the prediction. When the target, a vehicle in our case, is moving at high speed, this uncertainty is small in the direction perpendicular to the moving direction, and high in the direction parallel to the moving direction. This is because vehicles cannot make a quick turn when undergoing fast motion. On the other hand, when a vehicle is moving at a slower speed, the uncertainty is high in all directions, because the vehicle can make a sharp turn in any moment. Therefore, we use a velocity-dependent covariance, which is omnidirectional at slow speeds and unidirectional at high speeds.

3.2. Tracker Correspondence

The key idea of our proposed tracking algorithm is the use of two trackers that run in parallel and complement each other. For this to work, we need to have a correspondence between targets from each tracker. We determine this correspondence by comparing the trajectories of tracklets from the background subtraction based tracker with trajectories of tracks from the regression tracker. More specifically, we match a tracklet with a track that has the maximum overlap with it. If this overlap is greater than some threshold, and if the tracklet and track are not moving in opposite directions, they are associated together.

Once we have this correspondence, we do a "reconciliation" step in every frame where we determine for each target whether the regression tracker has lost track and needs to be updated with the associated tracklet. We do this by first checking whether the two trajectories of a track and its corresponding tracklet are diverging. This is determined by measuring the amount of overlap over time. If the overlap decreases in every frame by more than a threshold, the trajectories are deemed to be diverging, and we need to determine which tracker to trust more. If the trajectories are converging, no further action is necessary.

To determine which tracker to trust more, we generate a feature vector containing tracker confidence values from both trackers. The confidence value for observations estimated by the detection-based tracker is the max-marginal probability (belief) resulting from MAP inference in [17]. The confidence values for observations estimated by the regression tracker are the probability from the mixture-of-Gaussian distribution on line 10 of Algorithm 1, as well as a likelihood value returned from a simple blob appearance model based on a center-surround feature (not discussed here). This feature vector is then used in a logistic regression classifier to make a decision which tracker to trust more. This classifier is trained offline with manually labeled examples of diverging trajectories. The labels are binary, indicating which trajectory is the correct one.

If the regression tracker is classified to be less trustworthy than the detection based tracker for a particular target, we initialize a new regression model from the tracklet, and add it to the list of regression models maintained by the regression tracker. In other words, we only update the appearance model in the regression tracker when it fails, which brings in significant computational savings. When a regression tracker has multiple regression models, Algorithm 1 is applied for each, and the regression model with the smallest determinant on line 8 of the Algorithm is chosen. We limit the number of regression models to 10 most recent. We also reset the motion model in the regression tracker to follow the motion of the tracklet.

4. Results

We evaluated the proposed algorithm on a sequence from a publicly available wide area aerial imagery dataset [1]. The dataset was mosaicked [16], stabilized [16], and georeferenced prior to tracking. We selected a 1408×1408 (429 m $\times$ 429 m) region for evaluation (see [9]). The dataset provides ground truth for 1025 frames. The selected region has about 24 targets in every frame (410 tracks in total), and several places where cars stop and start.

Evaluation Metrics. We examined the algorithm's detection and tracking performance. Detection performance was characterized by: *Detection Rate/Recall* (the fraction of detections in the ground truth found in the estimated tracks), *Precision* (the fraction of detections in the estimated tracks found in the ground truth), *False Positives per Frame*, *False Positives per Ground Truth* (average number of false positives for every ground truth detection), and *MODA* (Multiple Object Detection Accuracy, see N-MODA in [18]). Tracking performance was characterized by: *Track Swaps* (average number of ID-switches in a ground truth track), *Track Breaks* (average number of times a ground truth track is not tracked from one frame to the next), and *MOTA* (Multiple Object Tracking Accuracy, where $c_s = 1$ [18]). Computational efficiency was measured by the average number of frames processed per second, but only for C++ code.

Setup. We used a detector based on background subtraction to provide detections to the algorithm. Since the detector can be configured in many ways to achieve different precision and recall performance, we performed the evaluation with two representative configurations. In one configuration, the detector has an F_1 score of 0.13, and in the other one, the detector has an F_1 score of 0.21. Both configurations were fully evaluated using the metrics above, allowing one to better understand the context of the tracking performance.

We compared our approach to [17], [15], and [19]. All of the approaches used the same detector configuration, and for all approaches, estimated tracks shorter than 5 frames were removed from evaluation (considered to be mostly unreliable). A window size of 8 frames was used in [17].

Evaluation of [15] was based on a modified version of the publicly available MATLAB code provided by its authors. All detections were given equal cost, and other costs were set such that only tracks of at least 5 frames would be generated. Detections were linked if the distance between them was less than 60 pixels (18 m), and they were from adjacent frames (no occlusion allowed). Adding edges between detections from frames further apart did not help.

Evaluation of [19] was based on our own implementation of the algorithm with no gradient suppression. We deliberately did not include gradient suppression to keep the set of target detections the same in all approaches, and evaluate the tracking (data association) performance. We set $p = 0$, not allowing any occlusions, as doing otherwise did not improve the result.

Discussion. The results are shown in Table 1, and more results are available on our website [9]. The object detection rate of the proposed approach is higher than that of the competing methods, which are purely detection based. This indicates that the proposed approach, which combines a detection-based tracker with a frame-to-frame regression tracker, makes a better use of the available set of detections to track targets longer than the competing methods.

The precision of the proposed approach is less than that of [17], but still beats [15] and [19]. As described, these approaches do not model target appearance, and rely on the detector to provide a decent set of detections. When the detector is noisy, as in our experiments, these approaches find it difficult to determine which detections are false. Since the resolution of targets in our imagery is limited, it is not clear how much would the precision improve if appearance modeling had been implemented there. Motion modelling is also limited in different ways in [15] and [19].

The proposed approach is also a clear leader in tracking accuracy, as measured by the number of ID-switches and track breaks. The average number of ID-switches does not change at all with different detector configurations, which indicates the robustness of our approach. Notice that as the detector gets noisier (higher recall, lower precision), the proposed approach's MOTA does not drop, and actually increases a bit, in contrast to [15] and [19]. This is primarily because the increase in false detections is not as high as in [15] and [19], while the increase in recall is just as much, and the average number of ID-switches stays constant.

The main limitation of the proposed approach is increased computation time, which we are currently addressing. Also, while the approach clearly handles stopping targets better, some failures remain. One of the reasons is the initialization of the regression model. The regression model is initialized using detections from background subtraction, and when a target is moving slowly, the shape mask is going to be incorrectly estimated. With the wrong shape mask, our template based features do not work as well, and regression performance is degraded. Another reason is the failure of the track correspondence or reconciliation procedure to correct the failing regression model of a track with a detection based tracker.

5. Conclusions

Current approaches for multiple target tracking in wide area imagery often rely on background subtraction, which is noisy and does not provide detections when targets stop. We have presented a multiple target tracking approach that does not have this complete reliance and is better able to track targets through stops, bringing us closer to persistent tracking. Results on wide area surveillance imagery show increased detection rates, and decreased ID-switches with limited increases in false alarms. In the future, we plan

	Tracking performance with detector F_1=0.13					Tracking performance with detector F_1=0.21				
	Proposed	[17]	[15]	[19]	Det. only	Proposed	[17]	[15]	[19]	Det. only
Detect. Rate (Recall)	0.48	0.41	0.36	0.44	0.57	0.43	0.38	0.32	0.40	0.50
Precision	0.92	0.97	0.14	0.14	0.08	0.95	0.98	0.25	0.28	0.13
False Pos. per Frame	1.03	0.35	53.5	65.1	163	0.54	0.19	23.6	24.7	80.5
False Pos. per GT	0.04	0.01	2.23	2.72	6.81	0.02	0.01	0.99	1.03	3.35
MODA	0.44	0.39	-1.87	-2.27	-6.25	0.41	0.37	-0.66	-0.63	-2.85
Track Swaps	0.20	0.36	1.23	1.31	-	0.20	0.19	0.86	0.51	-
Track Breaks	0.99	1.77	2.80	3.10	-	0.92	1.79	2.13	2.00	-
MOTA	0.43	0.39	-1.90	-2.30	-	0.41	0.37	-0.68	-0.63	-
Frames per Second	0.12	19.3	-	3.80	-	0.12	33.1	-	9.07	-

Table 1. Comparison of the proposed approach with competing methods for different detector configurations. The proposed approach has the highest recall, and the lowest average number of ID-switches and track breaks. The false alarm rate is low, only slightly worse than [17]. Its primary limitation is the increased computational cost. '-' indicates unavailable or irrelevant values.

to decrease the computational complexity of the proposed approach and investigate joint tracking of multiple targets with explicit constraints between them, which would likely be important in dense tracking scenarios.

References

[1] AFRL. WPAFB 2009. `https://www.sdms.afrl.af.mil/index.php?collection=wpafb2009`. 6

[2] S. Avidan. Ensemble tracking. *IEEE Transactions on PAMI*, 29(2):261–271, 2007. 3

[3] B. Babenko, M.-H. Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *IEEE CVPR*, pages 983–990, 2009. 3

[4] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *IEEE Transactions on PAMI*, 33(9):1806 –1819, sept. 2011. 1, 2

[5] T. B. Dinh, N. Vo, and G. Medioni. Context tracker: Exploring supporters and distracters in unconstrained environments. In *IEEE CVPR*, pages 1177–1184, 2011. 5

[6] G. Doretto and Y. Yao. Region moments: Fast invariant descriptors for detecting small image structures. In *IEEE CVPR*, pages 3019–3026, 2010. 2

[7] S. Hare, A. Saffari, and P. H. S. Torr. Struck: Structured output tracking with kernels. In *IEEE ICCV*, 2011. 2, 3

[8] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001. 5

[9] J Gradient. Persistent Tracking for Wide Area Aerial Surveillance. `jgradient.com/pubs/prokaj.cvpr14/`. 7

[10] Z. Kalal, J. Matas, and K. Mikolajczyk. P-n learning: Bootstrapping binary classifiers by structural constraints. In *IEEE CVPR*, pages 49–56, 2010. 3, 5

[11] M. Keck, L. Galup, and C. Stauffer. Real-time tracking of low-resolution vehicles for wide-area persistent surveillance. In *IEEE WACV*, pages 441–448, 2013. 1, 2

[12] P. Liang, G. Teodoro, H. Ling, E. Blasch, G. Chen, and L. Bai. Multiple kernel learning for vehicle detection in wide area motion imagery. In *International Conference on Information Fusion (FUSION)*, pages 1629–1636, 2012. 2

[13] P. J. Moreno, P. P. Ho, and N. Vasconcelos. A kullback-leibler divergence based kernel for svm classification in multimedia applications. In *Advances in NIPS 16*. 2004. 5

[14] A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and W. Hu. Multi-object tracking through simultaneous long occlusions and split-merge conditions. In *IEEE CVPR*, volume 1, pages 666–673, 2006. 1, 2

[15] H. Pirsiavash, D. Ramanan, and C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *IEEE CVPR*, pages 1201 –1208, 2011. 1, 2, 7, 8

[16] J. Prokaj. *Exploitation of Wide Area Motion Imagery*. PhD thesis, University of Southern California, 2013. 3, 6

[17] J. Prokaj, M. Duchaineau, and G. Medioni. Inferring tracklets for multi-object tracking. In *IEEE CVPRW (WAVP)*, pages 37–44, 2011. 1, 2, 3, 6, 7, 8

[18] R. Kasturi, D. Goldgof, P. Soundararajan, et al. Framework for performance evaluation of face, text, and vehicle detection and tracking in video: Data, metrics, and protocol. *IEEE Transactions on PAMI*, 31(2):319–336, Feb 2009. 7

[19] V. Reilly, H. Idrees, and M. Shah. Detection and tracking of large number of targets in wide area surveillance. In *ECCV*, volume 6313 of *LNCS*, pages 186–199. 2010. 1, 2, 7, 8

[20] X. Shi, H. Ling, E. Blasch, and W. Hu. Context-driven moving vehicle detection in wide area motion imagery. In *ICPR*, pages 2512–2515, 2012. 2

[21] A. Thayananthan, R. Navaratnam, B. Stenger, P. Torr, and R. Cipolla. Multivariate relevance vector machines for tracking. In *ECCV*, vol 3953 of LNCS, pages 124–138. 2006. 6

[22] O. Williams, A. Blake, and R. Cipolla. A sparse probabilistic learning algorithm for real-time tracking. In *IEEE ICCV*, volume 1, pages 353–360, 2003. 2, 3, 4, 5

[23] J. Xiao, H. Cheng, H. Sawhney, and F. Han. Vehicle detection and tracking in wide field-of-view aerial video. In *IEEE CVPR*, pages 679 –684, 2010. 2

[24] L. Zhang, Y. Li, and R. Nevatia. Global data association for multi-object tracking using network flows. In *IEEE CVPR*, pages 1–8, 2008. 2