EXPLOITATION OF WIDE AREA MOTION IMAGERY

by

Jan Prokaj

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

August 2013

Dedication

To my family. Ďakujem Vám za Vašu nekonečnú podporu.

Acknowledgments

This dissertation would not have seen the light of day without the guidance of my advisor Professor Gérard Medioni. Thank you for your patience and support, especially when I was not doing so well.

Thank you Professor Ram Nevatia for agreeing to serve on my guidance committee, for your willingness to serve on my dissertation committee at the last minute, as well as for your comments.

Thank you Professor Shri Narayanan for agreeing to serve on my dissertation committee and for your comments.

Thank you Professors John Wilson and Fei Sha for agreeing to serve on my guidance committee and your advice.

Thank you Sheila Vaidya, Lawrence Livermore National Lab, for funding my Ph.D. study.

I would also like to thank my fellow IRIS lab members that I got a chance to interact with and that made my stay at USC more enjoyable. Special thanks to Xuemei Zhao and Dian Gong. Thanks for the pizzas, wine, real Chinese food, and so much more.

Table of Contents

List of Tables

List of Figures

Abstract

Current digital photography solutions now routinely allow the capture of tens of megapixels of data at 2 frames per second. At these resolutions, a geographic area covering a whole city can be captured at once from an unmanned aerial vehicle (UAV), while still allowing the recognition of vehicles and people (for sensors under development). This fact, in tandem with the availability of increased computational power, has led to the growth of *wide area motion imagery (WAMI)*.

This imagery opens doors to a lot of applications, for example, in urban planning, security, and geospatial digital libraries. However, we have found out that this imagery cannot be easily processed by human operators, just because of its sheer amount. Therefore, we propose that a paradigm shift is required to enable working with UAV video data. The task of visualization cannot be simply defined as looking at the imagery. In reality, the pixel stream is a signal rich with information that must be extracted into a form that is easily processed by a person. The development of this extraction process falls into the realm of computer vision and is the heart of this dissertation.

WAMI data is often captured by an array of cameras. Therefore, at the lowest level, we need an algorithm that takes an array of individual camera images and estimates a high quality

mosaic. We propose a piecewise affine model to handle all image deformations that deviate from the standard pinhole camera model. The results show our model produces better mosaics than a standard lens distortion model.

Detection of moving objects is not possible with a moving camera. Video stabilization transforms the mosaicked video stream into one without any camera motion. We propose a stabilization technique which minimizes the amount of drift and jitter in the stabilized imagery. The results show our technique has better accuracy compared to widely used techniques.

The next level of processing involves estimating the trajectories of all moving objects, or "tracking." We propose a tracking algorithm that optimally infers short tracks using Bayesian networks. These tracklets are then integrated into a multi-object tracking algorithm that achieves good performance on aerial surveillance video. When coupled with a regression-based tracker, stopping targets can be handled.

WAMI is often collected over urban areas, where there are tall buildings, and other structures, which cause severe occlusion that in turn causes significant track fragmentation. To solve this problem, we propose a method which links fragmented tracks using known 3D scene structure. Our method outperforms the classic Hungarian algorithm.

In order to enable large scale semantic analysis of WAMI data, higher level algorithms that determine at least some of the semantics are necessary. Expecting full semantic description of the scene is unrealistic, but the task can be made easier for a human operator by automatically determining the most common, or primitive, events or activities. We propose a framework

based on the Entity Relationship Model that is able to recognize a large variety of activities on real data as well as GPS tracks.

CHAPTER 1

Introduction

People have always had a large interest in keeping track of the environment and activities around them. This monitoring has been primarily driven by defense and security reasons. Understanding external forces allows one to predict them and take appropriate action to protect themselves. Early examples of these monitoring "systems" and behaviors are watch towers, and the construction of castles and forts on hills. One obvious disadvantage of these solutions is that it required the location of the sensor (usually a person) close to the external force of interest. This is a problem when it's in an area that is inaccessible, far from the nearest settlement, or dangerous (enemy territory, eye of the hurricane). It limits the amount of time for corrective action or risks the loss of a human life.

With the advent of photography and aviation, remote monitoring, or remote sensing, has become possible. By mounting a camera (or other scientific instruments) on an aircraft, one can take a photograph of the world below from a high altitude, and gain the ability to observe

Figure 1.1: Some of the commonly used UAVs are Predator MQ-1 (left) and Reaper MQ-9 (right), both manufactured by General Atomics.

a large geographic area with low risk. Furthermore, this observation happens with a relatively little time delay for an arbitrary location in the world, on the order of several hours or a day. The introduction and advancement of satellite technology in the second half of the 20th century has made remote sensing a non-stop operation.

Another breakthrough in remote sensing has occurred recently with the development of unmanned aerial vehicles (UAVs). These are small aircraft, often called drones, which are piloted remotely, or flown semi-autonomously. They are much cheaper to deploy than satellites, have lower communication costs, can fly at high altitudes to minimize chance of detection, or low altitudes to avoid cloud cover, can produce imagery with various characteristics by flying in different flight paths or with different camera orientations, can easily change their payload (new sensor or a new weapon), and can fly for long periods of time. These properties have made them very popular for reconnaisance. Figure 1.1 shows examples of widely used UAVs.

Figure 1.2: At 0.40 meters per pixel, recent sensors can capture a city-scale area at few frames per second. This resolution makes it possible to observe and track movement of vehicles. However, computer vision algorithms are needed to automate the process, as it would take hundreds of analysts to monitor the imagery in real-time.

At the same time as UAVs were developed, research in optics and digital photography has made significant progress. Current digital photography solutions now routinely allow the capture of tens of megapixels of data at 2 frames per second. Sensors under development are able to capture more than 1 gigapixel images at similar frame rates. At these resolutions, a geographic area covering a whole city can be captured at once, while still allowing the recognition of vehicles and people (for sensors under development). See Figure 1.2 for an illustration. This fact, in tandem with the availability of increased computational power, has led to the growth of *wide area motion imagery (WAMI)*. It is also known as wide area aerial surveillance (WAAS) imagery, or full motion video (FMV), though the latter term is also used for reconnaisance videos that are not wide area. This imagery (actually a video) is characterized by its generation on aerial platforms, low temporal sampling rate, spatial resolution of 0.5 m/pixel or better, large format (tens of megapixels), and is in grayscale. Furthermore, physical constraints often

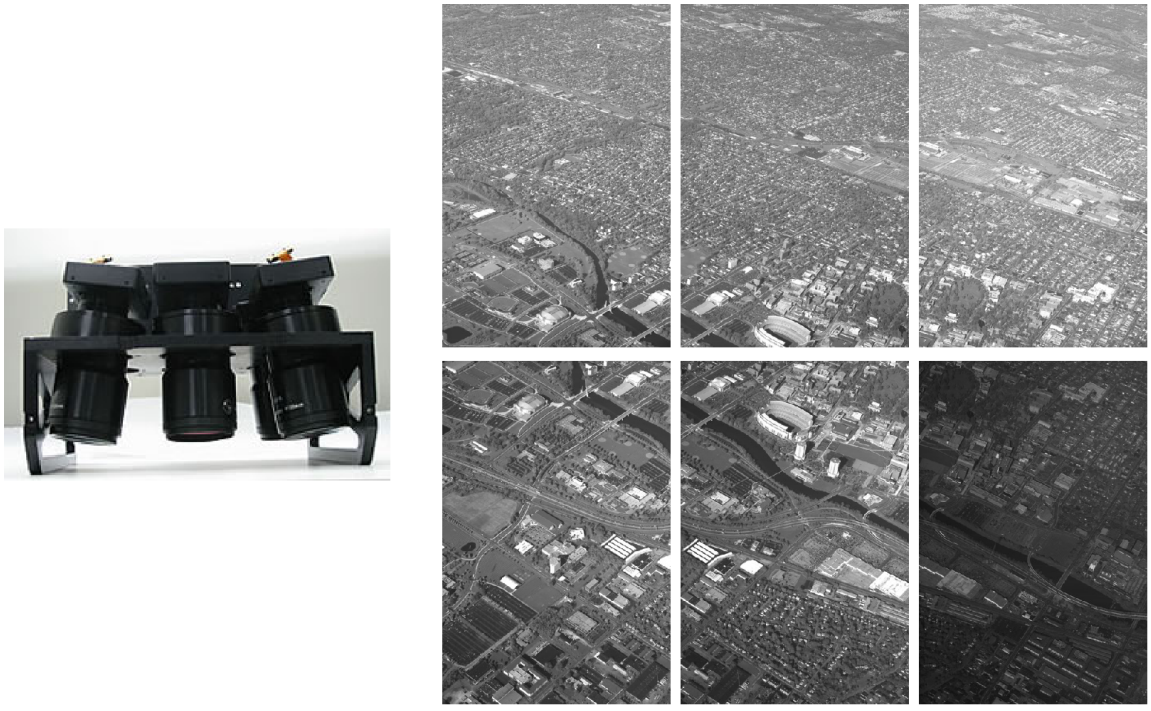


Figure 1.3: Wide geographic areas are captured by an array of sensors (cameras) from an aerial platform (left). As a result, every frame of data contains several images, one from each camera (right).

require that the imagery is captured by an array of smaller sensors sharing an optical center, rather than one large sensor. This is illustrated in Figure 1.3. All of these characteristics have non-trivial practical computational implications as will be discussed later.

1.1 Exploiting WAMI

Wide area motion imagery has given us what we wanted: the ability to continuously and in real-time monitor our world. This ability opens doors to a lot of applications, for example:

- **Urban Planning.** By tracking the movement of vehicles throughout the day, inefficiencies in the road network or traffic signalling can be identified and removed, and public transportation system can be optimized to serve the greatest amount of people.

- **Security.** A model of normal traffic flow around locations of interest can be inferred, allowing the detection of unusual vehicle movement. Similarly, vehicle or human activities known to be potential threats can be automatically recognized.

- **Digital Earth.** Georeferenced temporal data can be automatically published to geospatial digital libraries, creating a richer picture of the world. For example, one can lookup the change of geographic features (cities, waterways, highways, etc.) over time, or infer social behavior at a location of interest from the movement of vehicles or people over time.

However, we have found out that the data to support these applications, this data which we were seeking the whole time and that we now have, cannot be easily processed by human operators. This is easily seen in the reconnaisance context. For a UAV with a relatively low-resolution video stream on the order of a few megapixels (1920x1080), there are already several ($\approx$ 6) people involved in the operation. On top of that, the video stream is merely used for real-time visualization to support decision making (called "situational awareness" in defense circles). This mode of operation is no longer suitable for WAMI data, which is up to a gigapixel in size. First of all, we cannot even display one frame of the captured video on standard-sized screens. Assuming a display with 1920x1080 resolution, a grid of 28 screens would be needed to display one 7680x7560 frame ($<$ 60 megapixels), and with each display measuring 24 inches

in width, the grid of screens would be more than 2 meters across. How many people would be necessary to watch 28 high definition monitors? For a frame of one gigapixel in size, more than 480 monitors would be necessary, and a wall measuring more than 13 meters across. Is it possible to integrate everyone's observation in every decision in this environment? Even if this were achieved, the big picture that the data provides, and which is so valuable, would be missed, because it would be very hard to "connect the dots" between an activity observed in one subregion with an activity in another subregion. Therefore, we propose that a paradigm shift is required to enable working with UAV video data. The task of visualization cannot be simply defined as looking at the imagery. In reality, the pixel stream is a signal rich with information that must be extracted into a form that is easily processed by a person. The development of this extraction process falls into the realm of computer vision and is the heart of this dissertation.

1.1.1 Problem Statement

The objective of this dissertation is to develop algorithms that automatically process WAMI imagery to turn it into a more useful, informative form. Depending on the intended application, this more informative form can exist at different levels of semantics, from low-level to high-level. Therefore, the set of algorithms we propose operates in range from low-level processing to high-level processing and is illustrated in Figure 1.4. As high-level tasks depend on the results from low-level tasks, the algorithms we propose operate in a pipeline.

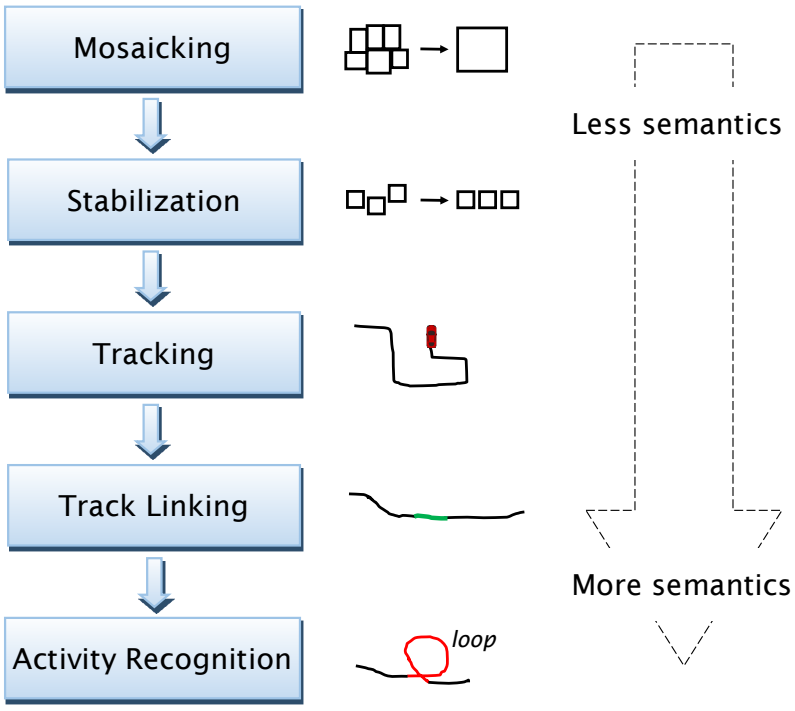


Figure 1.4: The process of extracting semantics from wide area motion imagery.

At the lowest level, we need an algorithm that takes an array of individual camera images and estimates a highly accurate mosaic. This mosaic image is more useful for higher levels of processing than the individual camera images, because it stores one frame of data in one piece and avoids camera hand-off logic that would be required otherwise for tracking. Once mosaicked, the video stream needs to be stabilized so that the motion of objects becomes apparent. During stabilization, the imagery is warped (transformed), such that the motion of the camera (airplane) is removed. The next level of processing involves estimating the trajectories of all moving objects, or "tracking." In our datasets, these objects are vehicles, but we are interested in all moving objects in general. This level is important, because moving objects tell us a lot about what is happening in the scene, and are a foundation for higher levels of processing, such as activity recognition. Track fragmentation should be minimized. Unfortunately,

in urban areas, tall buildings, and other structures, cause severe occlusion that in turn causes significant track fragmentation. This phenomenon is not specific to our tracking algorithm, but applies in general when the appearance of moving targets is limited, and their motion is non-linear (unpredictable). We would like an algorithm that mitigates this problem. Finally, in order to enable large scale analysis, higher level algorithms that determine at least some of the semantics are necessary. Expecting full semantic description of the scene is unrealistic, it is better to leave that up to the user, but the task can be made easier for him/her by automatically determining the most common, or primitive, events or activities. For example, answering the question "is this vehicle being driven safely?" is answered much quicker if the user has access to a list of speeding violations, or u-turns.

1.1.2 Challenges

Solving the problems above is essential for exploiting WAMI to support the desired applications. To do so, there are several computational issues and challenges to overcome:

- Mosaicking affects all subsequent processing (background model estimation, etc.) and needs to be high quality (no visible seams). At the same time, the mosaic should be physically possible to enable further computer vision analysis. When cameras vibrate, the parameters of the mosaic slightly change each frame and may need to be continuously re-estimated, requiring an effcient algorithm. Furthermore, each camera has different imaging characteristics, leading to large variation in intensity across the sensor array.

- Stabilized imagery should have minimal drift and jitter, which may be challenging to achieve when camera viewpoint changes significantly over time and there are few feature correspondences.

- The tracking algorithm needs to scale to a large number of targets while handling their small size. Small target size limits the complexity of appearance models, leading to increased ambiguity in tracking. Furthermore, parallax from tall structures often leads to erroneous moving object detections.

- The tracking algorithm also needs be able to track targets through stops. The commonly used method of background subtraction can not be used as exclusive means of object detection.

- Predicting the location of an occluded target becomes nearly impossible when occlusions get long and the target undergoes acceleration (makes turns, stops, etc.). More scene knowledge, such as the 3D structure is needed to resolve this problem, but estimating this from data is a challenge on its own.

- Activities may occur over different time scales, may manifest themselves as different sequences of events, and depend on uncertain tracks.

1.2 Outline

The contribution of this work is a set of algorithms for processing WAMI that attack the problems and challenges listed above. Mosaicking of an array of cameras is addressed in Chapter 2, where we also discuss a way to handle the inter-camera differences in bias and gain. In Chapter 3 we explain our method for video stabilization, followed by a multi-object tracking algorithm in Chapter 4. Chapter 5 couples this algorithm with a regression tracker to handle stopping targets. Robust occlusion handling in urban areas using known 3D scene structure and sequence alignment is detailed in Chapter 6. Activity recognition is described in Chapter 7. The papers serving as the basis for Chapters 2-7 are listed as follows:

- Chapter 2

  J. Prokaj and G. Medioni. Accurate efficient mosaicking for Wide Area Aerial Surveillance. In *Workshop on Applications of Computer Vision*, 2012.

- Chapter 4

  J. Prokaj, M. Duchaineau, and G. Medioni. Inferring Tracklets for Multi-Object Tracking. In *Workshop of Aerial Video Processing Joint with IEEE CVPR*, 2011.

- Chapter 5

  J. Prokaj and G. Medioni. Persistent Tracking for Wide Area Aerial Surveillance. In submission.

- Chapter 6

  J. Prokaj and G. Medioni. Using 3D Scene Structure to Improve Tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1337-1344, 2011.

- Chapter 7

  J. Choi, Y. Dumortier, J. Prokaj, and G. Medioni. Activity recognition in wide aerial video surveillance using entity relationship models. In *International Conference on Advances in GIS (SIGSPATIAL)*, 2012.

CHAPTER 2

Mosaicking

Physical limitations and cost in the manufacturing process prevent the production of a single large lens and image sensor. Therefore, wide area motion imagery data is often captured by an array of CMOS and cameras, as is illustrated in Figure 1.3. The number of sensors in the array can vary, but a common number is 6. Even though physically there is no one image of the scene, it is still desirable to generate one virtually, as if it were captured by a single camera. This is illustrated in Figure 2.1. Having access to such image, called a mosaic, has at least two benefits: it makes it easy to see the extent of the geographic area being captured and thus simplifies UAV control, it avoids the problem of tracking moving objects across several cameras, which adds algorithmic complexity. One potential disadvantage of this image is that if it is inaccurate (has visible seams), or physically unrealistic, errors will result in algorithms that use this image is input. For example, image stabilization will not be possible and there

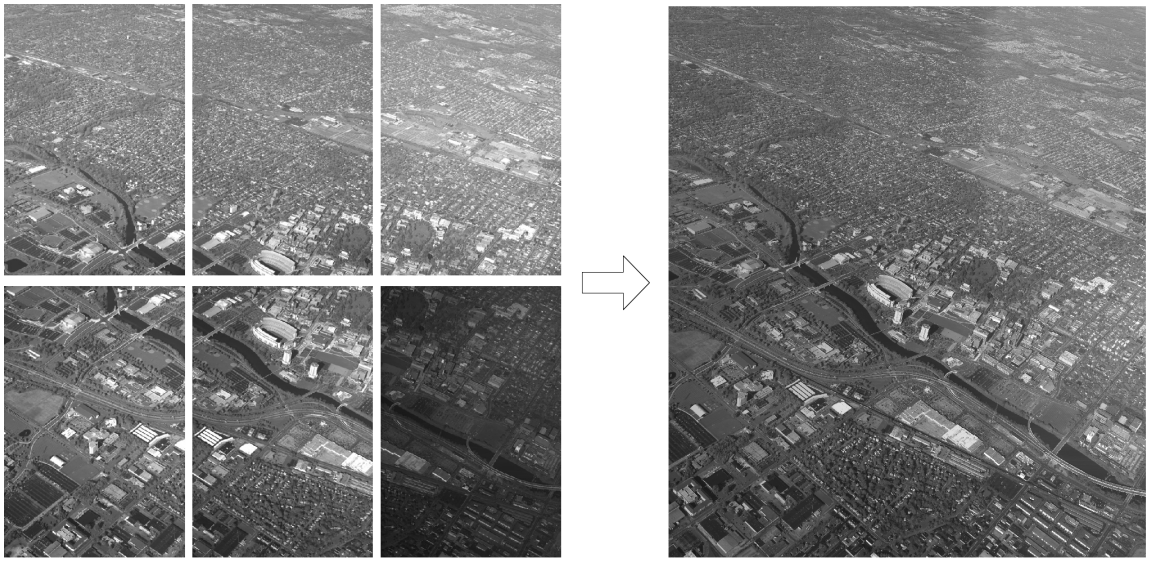


Figure 2.1: One frame of the imagery is captured by an array of cameras (left), while it is desirable to work with only one image per frame, as if it were captured by a virtual camera (right).

will be many false moving object detections around the seams. Consequently, high quality and physically realistic estimation is required.

Even though the cameras in the sensor array are fixed, there may be slight, but noticeable, changes in the sensor geometry at every frame. These may be caused by vibration or imperfections in mechanical mounting. This means that sometimes we cannot rely on a one-time, offline, estimate of the mosaic, but must re-estimate it online for every frame. Therefore, it is desirable that the estimation of the mosaic be efficient.

To facilitate the generation of this virtual image, the sensors in the array are arranged so that they all share one optical center (or center of projection). In other words, the 3D transformation from the pose of one sensor in the array to another is a rotation (there is no translation component). This helps, because it is well known [32] that if there is no translation between two cameras, the image transformation between them is fully described by a homography

$H = K^{'}RK^{-1}$, where R is a 3D rotation matrix, K' is the calibration matrix of the destination camera, and K is the calibration matrix of the source camera. By selecting one of the images in the array as a reference, and estimating homographies between it and the rest of the images in the array, all images can be "registered", or aligned to a common coordinate frame, creating our mosaic.

However, this geometry model is only valid for true pinhole cameras, which are not used in practice. There are additional factors such as lens distortion which corrupt this model. Therefore, a model with more degrees of freedom is necessary to accurately register the images. A common solution to this is to adopt a global lens (or radial) distortion model [80, 75, 74]. We have observed that in wide area imagery, this model does not accomodate all of the deviations from the pinhole model and an even more flexible model is needed. We propose a piecewise affine model to solve this problem. It is able to generate high-quality mosaics by allowing a non-rigid deformation of the image, and it is able to estimate this deformation efficiently by carefully selecting a small number of optimization constraints. The results show that our piecewise affine model has better accuracy compared to the classic radial distortion model used in previous work.

Geometrical alignment is not sufficient to produce a seamless mosaic. Notice in Figure 2.1 that there is a large variation in intensity across different cameras in the sensor array. Therefore, intensity alignment is also needed to generate a seamless mosaic. We choose to parameterize the intensity correction in each camera with a standard linear model (bias and gain). The estimated intensity corrections show good results.

2.1 Related Work

The interest in mosaic computation has a rich history. Some of the early applications include whiteboard or document scanning [84], video enhancement [58], video compression [37], and video indexing [36]. In all these applications, what is being mosaicked is a sequence of images taken from the same camera, which is in contrast to the nature of mosaicking done here, where the images to be mosaicked come from different sensors. This makes the mosaic estimation problem more difficult since different sensors potentially have significantly different camera calibrations and image response functions.

One of the major problems in mosaicking is global alignment of a set of images. Since homographies are estimated between pairs of images, to generate a mosaic which is generated from more than two images, several homographies may need to be concatenated (multiplied) to estimate a correct transformation from one image to the reference image. Each time two homographies are multiplied, a small error is introduced. After a few multiplications, the accumulated error is large enough to cause misalignment. This problem is addressed by [43, 79, 74]. The common solution is to optimize all homographies of the mosaic at once, with a prior topology calculation step if necessary [85]. In our application, the number and arrangement of images in the mosaic is known, therefore topology determination is not necessary. However, a joint optimization of the geometry is still needed to achieve the best result.

Deviations from the pinhole camera model in the context of mosaic construction are discussed in [80, 75, 79]. In [80, 75], a radial distortion model is adopted to handle all residual

error. Shum and Szeliski [79] propose a local alignment step based on optical flow that handles arbitrary camera model. However, this introduces too many degrees of freedom and the estimation of this model is more likely to get stuck in local minima.

A similar problem to mosaicking is video stabilization. The goal there is to take an existing jittery image sequence (usually from a hand-held camera) and use it to synthesize a new sequence without any jittering. Warping the input images to their stabilized configuration without any artifacts requires a non-rigid deformation of the input. Recent methods in this area [35, 55] have used a similar image deformation model as in this work, called as-rigid-as-possible deformation. However, estimating such deformation requires precise, uniformly distributed, correspondences between images. In our experience, the current state-of-art feature detectors and descriptors do not produce such correspondences.

Another approach to mosaicking was recently proposed by Lin *et al*. [54]. There, an affine stitching field is estimated, similar to our piecewise affine model. Instead of using point correspondences directly to estimate the stitching field, which causes problems as mentioned above, the authors formulate the problem as finding a smooth stitching field that minimizes the difference between the SIFT descriptors of the resulting (implied) correspondences. While the stitching results are state-of-art, the flexibility of the algorithm to fit generalized motion comes at a computational cost. A MATLAB implementation is able to stitch a pair of 500x500 images in 8 minutes. In our domain, we are interested in stitching 6 images, each about 4008x2672 in resolution, which is not tractable by their approach.

When a sequence of images is captured over time, each frame may be subject to different lighting from the environment. As a result, compensation by auto-exposure and other image processing control in the camera will cause variation in intensity in the captured images. This variation will manifest itself as visible seams in the generated mosaic, which is undesirable. A solution proposed by [47] is to estimate the camera response function and align the image sequence to a common exposure. This approach, however, requires estimating epipolar geometry and dense stereo correspondeces for every pair of images, which is not practical in our application. In [70], the intensity of each image is adjusted using the gamma function. The parameters of the function are estimated by minimizing the intensity difference for corresponding pixels between the image and a reference image.

Besides correcting for intensity variation, it may also be necessary to account for small registration errors or arbitrary image differences that cause seams in the mosaic. A classic solution to this problem is to blend the overlapping images. A multi-frequency blending method popularized by [11] that does not blur high-frequency content in the image is in [12]. Blending that minimizes inconsistency in the gradient domain was presented in [52]. While the results appear high quality, the extra computation cost is not worth the additional improvement over the method by Burt and Adelson [12]. Therefore, the blending method by Burt and Adelson is adopted in our work.

2.2 Approach

Our goal is to generate a mosaic, which is geometrically accurate (all images in the array are well aligned), physically realistic (equations from computer vision geometry hold), and photometrically consistent (no apparent variations in intensity). Geometrical accuracy is the most important criterion, and we address this first.

The model which registers one image onto another needs to have enough degrees of freedom to handle the image deformation. For a true pinhole camera, this model is a homography, which has 8 degrees of freedom. In practice, this model does not hold and must be augmented with additional transformations. A point $\mathbf{p}$ in one image is thus transformed to point $\mathbf{p}'$ in another image as:

$$L\left(\mathbf{p}^{'};\theta\right) = H \cdot L\left(\mathbf{p};\theta\right) \tag{2.1}$$

where H is a homography and L is the additional transformation parameterized by θ. L is non-linear in general and can be a flow field, or a radial distortion correction (it undistorts the image). Here we propose to define L as a *piecewise affine model* (PAM):

$$L(\mathbf{q}) = \sum_{k=1}^{K} \delta(m(\mathbf{q}) - k) A_k \mathbf{q} \tag{2.2}$$

where K is the number of pieces, $m(\mathbf{q})$ is a function that determines the corresponding piece for point $\mathbf{q}$, and A_k is an affine transformation. Each piece covers a unique region of R^2. In other words, the pieces define a tessellation of R^2.

We have defined the tessellation to be triangular as follows. The image is first divided into a $2N \times 2N$ grid of equal sized cells, with $N \geq 1$. Each cell is then divided into 2 triangles, such that a radial pattern is created. In other words, cells in the northwest and southeast quadrants are divided with an up-diagonal and cells in the northeast and southwest quadrants are divided with a down-diagonal. The goal of this construction is to facilitate the representation of radial distortions. The tessellation is illustrated in Figure 2.2 for $N = 2$.

The piecewise affine model is not parameterized by affine coefficients, but by the coordinates of each grid point after the deformation. This ensures that affine transformations in neighboring triangles agree with each other on their shared border. Let V_i be the known coordinates of grid point i before the deformation, and let V_i' be the coordinates of the same grid point after the deformation. Given the three correspondences of a triangle, $V_i \leftrightarrow V_i'$, $V_j \leftrightarrow V_j'$, $V_k \leftrightarrow V_k'$, an affine transformation for that triangle is estimated by solving a small linear system. Therefore,

$$\theta = [V_1' V_2' \cdots V_M'] \tag{2.3}$$

where $M = (2N+1) \times (2N+1)$. For example, in Figure 2.2 there are 25 grid points, giving 50 parameters.

2.2.1 Estimating PAM

The piecewise affine model, as other parametric motion models, may be estimated using a direct method where all pixels are used in optimization [9] or a feature-based method. An advantage of a feature-based method is that it is usually faster than a direct method. However,

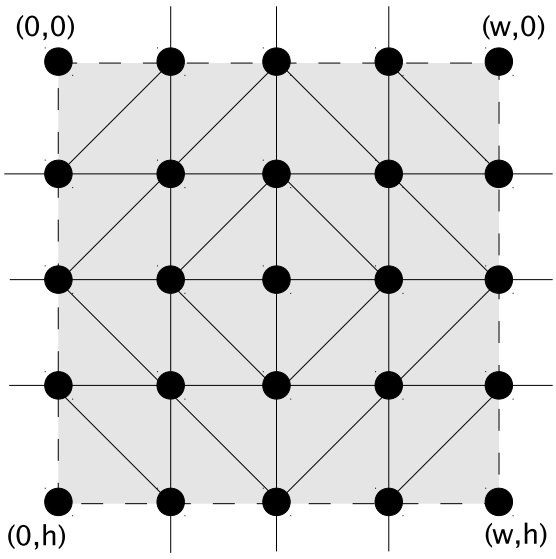


Figure 2.2: Tessellation of R^2 with respect to an image of width w and height h.

the disadvantage is that it requires correspondences to be uniformly distributed across the overlapping region, and the correspondences must be precise (low inlier noise) to guarantee a high quality mosaic. On the other hand, in a direct method, no correspondences are necessary. What is being optimized there is the intensity difference (or variance) of overlapping pixels. The disadvantage of this method is that it uses all overlapping pixels in the optimization, which can make the parameter estimation quite slow.

We initially experimented with a feature-based approach, but the results were not satisfactory. After switching to a direct method, we noticed immediate improvement in mosaic quality. Unfortunately, as expected, the computational burden was indeed a problem. In our domain, the number of overlapping pixels in a 6-camera sensor array is over 35 million! This number of pixels does not lend itself to real-time parameter estimation.

Baker *et al*. [6] made the important observation that not all pixels are necessary in this optimization. In fact, pixels with zero, or very small gradients have no effect on convergence.

By removing such pixels altogether, great computational savings are made. We have used a similar idea in our approach, but ensuring the pixels selected for optimization are uniformly distributed across the mosaic.

The mosaicking surface is first divided into a grid of $P \times P$ equal sized cells. We select one pixel from each cell, provided there is at least one pixel in the cell that overlaps two or more images. The pixel that gets selected is the one that has the highest Harris corner measure in one of the overlapping images. In other words, we select pixel p as

$$p = \arg\max_i \det(C_i) - \kappa \operatorname{tr}(C_i)^2 \tag{2.4}$$

where C_i is the second-moment matrix of pixel i. By selecting pixels in this manner, we ensure that all parts of the mosaic are aligned well and that each selected pixel makes as much contribution to the optimization of parameters as possible.

When using a direct method, an initial solution of the parameters is required. The parameters of L are initialized so that PAM is an identity map. The parameters of H are initialized by finding corresponding points in the images, using SIFT [57] for example, and using a robust RANSAC estimator. When mosaicking more than two images, the homographies estimated are homographies to the reference image. The reference image can be an arbitrary image in the array, but to keep image deformations to a minimum, it is best to select an image in the center. For images that do not overlap with the reference image, a homography is estimated by concatenating appropriate neighboring homographies. The resulting homographies to the

reference image are then jointly optimized to minimize the distance between corresponding points on the mosaicking surface, as expressed in this error function:

$$E(H_1, \cdots, H_I) = \sum_{\mathbf{p}_i, \mathbf{p}_j} ||H_j \mathbf{p}_j - H_i \mathbf{p}_i||_2^2 \tag{2.5}$$

where I is the number of images, H_i is a homography from image i to the reference image and $\mathbf{p}_i, \mathbf{p}_j$ are a pair of corresponding points from images i and j.

Given this initial solution, the parameter estimation proceeds in a coarse-to-fine fashion. Since our initial solution is often quite accurate, we build an image pyramid with only 3 levels. At the coarsest level, we minimize the intensity variance of selected pixels over the homography parameters. At the next two finer levels, we alternate between estimating the homography parameters and PAM parameters. The number of times we alternate is determined dynamically by measuring change in the parameters. When the root mean square (RMS) of the parameter difference vector is less than $1e-7$, we determine that we have converged. Homography parameters are optimized first, keeping the PAM parameters fixed. Then we minimize the intensity variance over the PAM parameters, keeping the homography parameters fixed. Formally, we minimize the following objective:

$$E(H_1, \cdots, H_I, \theta) = \sum_{\mathbf{p}} \frac{1}{O(\mathbf{p})} \sum_{o}^{O(\mathbf{p})} \left(I_o \left(L_o^{-1} \left(H_o^{-1} \mathbf{p} \right) \right) - \mu(\mathbf{p}) \right)^2 \tag{2.6}$$

where $\mathbf{p}$ is one of the selected pixels on the mosaicking surface, $O(\mathbf{p})$ is the number of overlapping images at pixel $\mathbf{p}$, $I_o(\mathbf{x})$ is the intensity of image o at location $\mathbf{x}$, and $\mu(\mathbf{p})$ is the

mean intensity of the overlapping pixels. Note that taking the inverse of PAM (L^{-1}) is more expensive than taking the inverse of a simple homography. In practice, most of the image deformation is handled by the homography, so that it is sufficient to use small grid sizes in PAM (such as 2x2 or 4x4), which does not make the inverse computation a significant problem. Even with the additional cost, the model still fares better compared to a radial distortion model, which cannot be inverted analytically. When an inverse of the radial distortion function needs to be computed, one either needs to compute a table (map) between undistorted and distorted coordinates and interpolate, or do iterative optimization to find the coordinates. Both choices are computationally expensive.

We minimize the objective using Levenberg-Marquardt algorithm [56]. This algorithm is an improved variant of gradient descent, and will converge to a local minimum. Unfortunately, this local minimum is not guaranteed to be near the global minimum, nor is it guaranteed to be a physically possible solution (there is no such constraint on the deformation parameters). In order to solve the first problem of not getting a global minimum, we must ensure a good initialization. We do this by estimating the parameters hierarchically, that is, using an image pyramid. We use 3 levels of the pyramid, but more can be used if necessary. However, the number of overlapping pixels decreases with more levels, and care must be taken there is enough of them to minimize the objective function.

To solve the second problem, and ensure we have a physically possible solution, we estimate the parameters using multiple frames of data and constrain the homography and PAM parameters to be fixed over time (same in every frame). Furthermore, we also assume the PAM

parameters are the same for each camera (θ does not vary across the camera array). This is not necessarily the case, but it seems unlikely that each camera in the array would be manufactured differently. Making these additional constraints decreases the number of parameters we need to estimate and improves the likelihood of reaching a global minimum. Fixing the homographies over time ensures that we get a physically possible solution under the assumption that the cameras do not shift or vibrate. We evaluate the validity of this assumption in our experiments. The objective function does not change with the introduction of multiple frames of data and additional contraints. The only thing we need to do is add pixels into the objective function from multiple frames of data (multiple mosaicking surfaces). The parameters will then be estimated such that the error is minimized across multiple frames.

2.2.2 Intensity correction

Geometrical alignment is not sufficient to produce a seamless mosaic. Notice in Figure 2.1 that there is large variation in intensity across different cameras in the sensor array. Therefore, intensity alignment is also needed to generate a seamless mosaic. There are various approaches to do this in the literature, such as [47]. Here we use a simple and fast approach that works well in practice to remove gross intensity differences.

We parameterize the differences in intensity as differences in camera gain (scaling factor). In other words, the intensity corrected image $I^{'}$ is a scaled version of the input image I:

$$I^{'} = gI \tag{2.7}$$

where g is an unknown gain correction factor. Therefore, we wish to estimate gain correction factors for each camera, such that the differences in intensity in the overlapping regions are minimized. It is easy to compute a correction factor for image A that optimally minimizes the intensity difference from image B. However, such an estimate is likely to conflict with a correction factor for the same image that minimizes the intensity difference from image C. It is clear the correction factors must be estimated jointly.

The pairwise correction factors provide a useful set of constraints for joint optimization. Consider the set of equations that must hold for optimal gain correction factors. Each pair of overlapping images A and B generates an equation such as this

$$g_A I_A = g_B I_B \tag{2.8}$$

$$\frac{g_A}{g_B} I_A = I_B \tag{2.9}$$

where g_A and g_B are unknown. For the same pair of images, we have

$$g_{AB} I_A = I_B \tag{2.10}$$

where g_{AB} is the known optimal pairwise correction factor. This implies that we have

$$g_A - g_{AB} g_B = 0 \tag{2.11}$$

for each pair of images. Stacking these equations together creates a linear system, whose non-trivial solution is the optimal set of gain correction factors, up to scale. We find the appropriate scale by assuming that the average of the corrections factors should equal 1. This minimizes the amount of intensity correction from input. It is necessary that the number of overlapping pairs of images be at least the number of images, but this is always satisfied for sensor arrays in our domain.

This intensity correction is performed before geometric alignment. To determine which pixels are in the overlapping regions, we use the initial solution of mosaic geometry from SIFT correspondences. Knowing the optimal solution is not necessary here. After geometric alignment, the images are blended using [12] to remove any remaining seams resulting from residual photometric differences or alignment errors.

2.2.3 Complexity

Parameters of PAM are estimated using non-linear optimization. The runtime depends on the number of iterations, and the complexity of each iteration. In each iteration, every sample is transformed to all the overlapping images and variance of the intensity at that location is estimated. Therefore, the complexity of each iteration is $O(SK)$, where S is the number of samples and K is the number of overlapping images. Note that $S \in O(CP^2)$, where C is the number of frames used in the optimization. When using an analytical Jacobian, the number of iterations in each round of optimization is less than 20 on average.

For intensity correction, we first estimate the pairwise gain correction terms and then solve a small linear system using SVD. Estimating the pairwise gain correction terms is $O(MWH)$ where M is the number of overlapping pairs of images and W, H is the width and height of each image. In common sensor array configurations, $M < 2K$, where K is the number of images. The size of the linear system is $M \times K$, which can be solved in $O(K^3)$ time.

We implemented the algorithm just presented in C++. Our implementation can estimate the parameters of a mosaick of 6 WAMI images using 14 frames of data (84 total images used in optimization) in 178 seconds for a model with $N = 1$ and $P = 450$ on an AMD FX-6300 processor. With 8 frames of data (48 total images used in optimization), the optimization time is reduced to 48 seconds.

2.3 Results

The proposed algorithm was evaluated on the Wright-Patterson dataset [51], which was captured by an array of 6 cameras. Each of the six images is grayscale and 4872x3248 in size. The accuracy of mosaicking was quantified using (2.6), except that all overlapping pixels were used in the evaluation. In the following discussion and plots, this number is divided by the number of pixels to give an average variance of intensity. No ground-truth geometry of the camera array was supplied by the dataset. Therefore, we can only measure the accuracy indirectly through (2.6). We make a reasonable assumption that the lower the objective value, the better the accuracy.

We evaluated several parameters of the proposed approach. These include the sampling grid size P (number of samples), model complexity N, the number of frames needed in the optimization, the difference between having fixed all parameters and allowing the homography parameters to vary, and physical plausability, as measured by stabilization accuracy.

To determine the appropriate number of samples needed to robustly estimate the mosaic, we took an empirical approach. We measured the mosaicking accuracy with respect to the different numbers of samples. Note that the number of samples is proportional to the sampling grid size P. Usually, the number is $< P^2$, since many of the cells do not contain any pixels with overlapping images. In a dataset similar to the one used for evaluation, a value of $P = 100$ corresponds to approximately 2762 samples per frame, $P = 300$ corresponds to 23568 per frame, and $P = 450$ to 52316 per frame. In our evaluation, three frames were randomly selected from the Wright-Patterson dataset, and mosaicking performance was measured there for each value of P. For this experiment, 8 frames from the dataset were used to estimate the parameters. These frames were uniformly distributed around the UAV's circular flight path. The resulting plot can be seen in Figure 2.3 for when $N = 1$. The dotted lines in the plot are a linear model obtained by a least squares fit to the data. We also observed a similar behavior when $N = 2$, although in that case the more complex model exhibited more variance, an indication that it is more susceptible to falling into local minima. The plot shows that in general, more samples lead to better accuracy. From this plot, we determined that an appropriate value of P to guarantee the best accuracy is 450. More or less samples can be

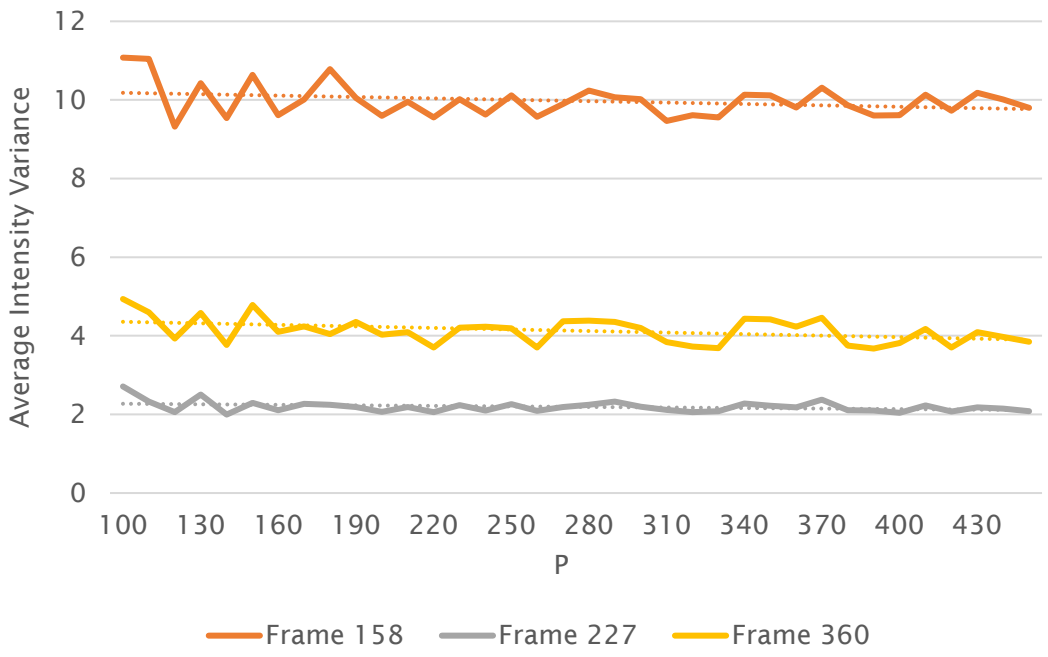


Figure 2.3: Effect of the sampling grid size P on mosaicking accuracy. In general, the more samples are used, the better the accuracy.

selected to tradeoff accuracy and computational cost. In any case, the number of samples is a small fraction of the total number of overlapping pixels, which is in the millions.

Similarly, to determine the appropriate model complexity (value of N) needed to robustly estimate the mosaic, we also took an empirical approach. For the first 300 frames of the Wright-Patterson dataset, we measured mosaicking accuracy for different values of N. The parameters were estimated using 14 uniformly distributed frames along the flight path. This plot can be seen in Figure 2.4. The different models produce virtually identical mosaics, with the more complex model faring slightly worse. This is an indication that the energy function of the more complex model is more non-linear than that of the simpler model, and there is

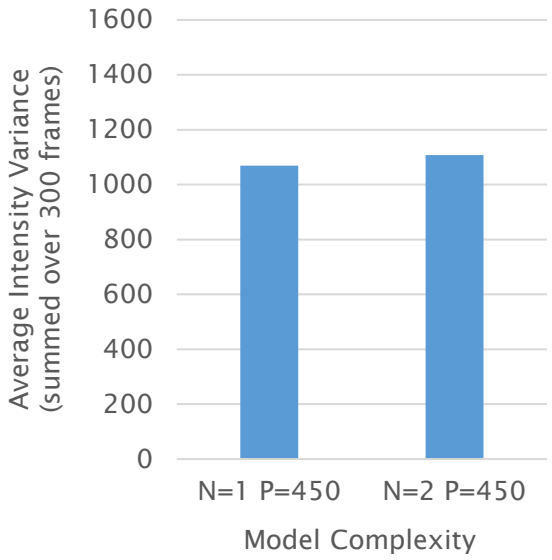


Figure 2.4: Effect of PAM complexity on mosaicking accuracy.

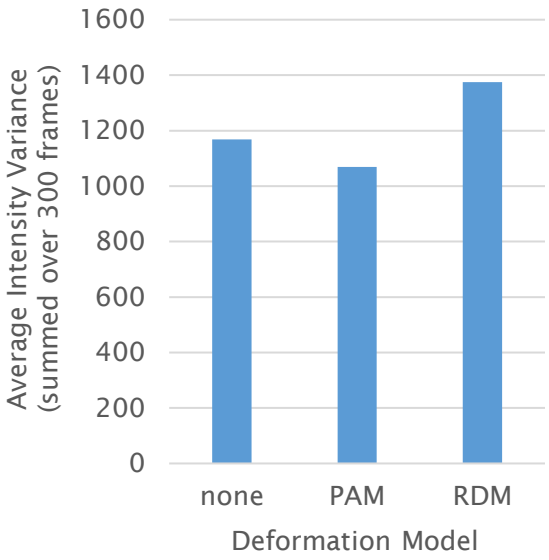


Figure 2.5: Comparison of different deformation models.

a higher chance of reaching a local minimum rather than a global minimum. Therefore we prefer the simpler model.

To determine the number of frames needed in the optimization, we estimated the mosaic parameters using different numbers of frames, and measured the accuracy for each. This plot can be seen in Figure 2.6. In general, the more frames are used in the optimization, the better the accuracy, although there is an asymptotic limit. With a small number of frames, the optimization is more likely to reach a local minimum or "overfit" to the frame(s) in the training set. The plot shows at least 8 frames should be used in the optimization for robustness.

We compared our approach with competing approaches [80, 75, 74] that use the radial distortion model (RDM) for non-rigid deformations. Our implementation models 2 radial distortion coefficients and 2 tangential distortion coefficients. In many of these approaches, the objective function is minimized using all overlapping pixels. In the dataset we used for

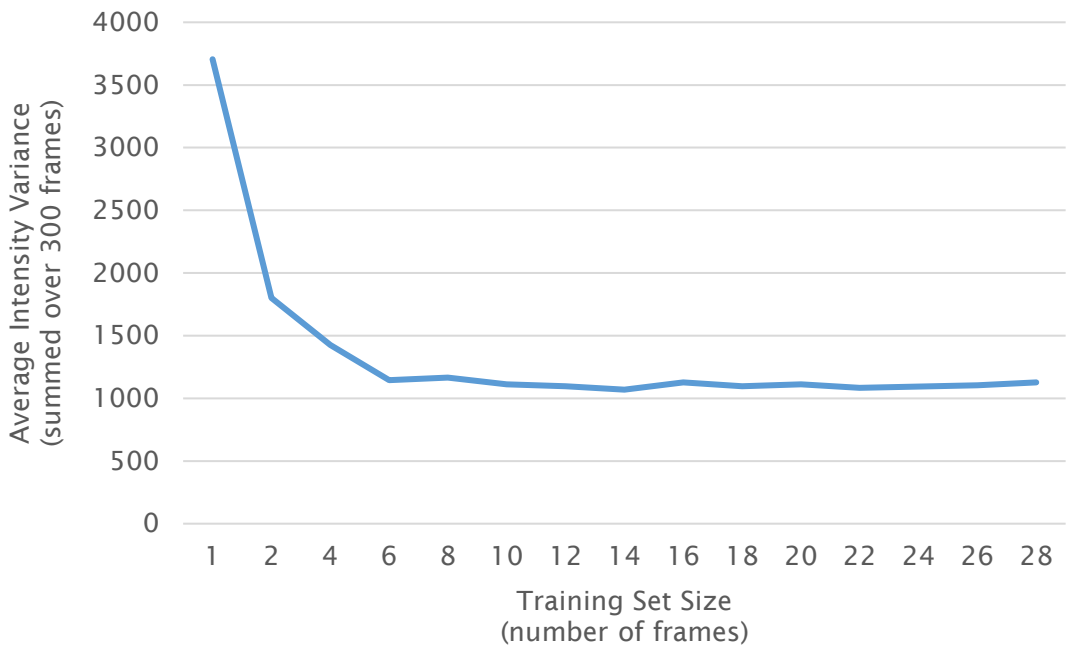


Figure 2.6: Effect of training set size on mosaicking accuracy. The larger the training set, the better the accuracy, up to an asymptotic limit.

evaluation, there are tens of millions of such pixels, which makes using all of them intractable. Therefore, in our implementation, we used our proposed method of carefully selecting pixels for optimization based on spatial gradients. Similarly for the baseline comparison with no non-rigid deformation (homography only). Quantitative comparison with the radial distortion model as well as the baseline is illustrated in Figure 2.5. The proposed piecewise affine model is able to achieve the best accuracy. The radial distortion model in our implementation actually produces the worst result, which is a bit unexpected. This is either because the radial distortion model is not the appropriate choice for this imagery, or it is just more difficult to estimate using our direct-method. Even if the RDM produced a good result, it is very expensive to estimate,

because there is no analytical inverse for the radial distortion function. Therefore, it is also unappealing from a computational cost perspective.

In our estimation of the mosaic parameters we assumed the cameras in the array do not move over time, and solved for only one set of homographies that is applied every frame. The validity of this assumption was evaluated in the next set of experiments. First, we used the fixed mosaic parameters estimated using 8 images as an initial solution, and optimized homography parameters every frame, while keeping the PAM parameters fixed. Allowing the homography parameters to vary over time this way effectively allows camera movement. We then compared the mosaicking accuracy of the PAM model with fixed homography parameters against the PAM model with varying homography parameters. This comparison can be seen in Figure 2.7. The plot shows that the PAM model with varying homography parameters is significantly more accurate (except for the first 13 frames, where we believe the optimization reached a local minimum). This is not terribly surprising, since there are more degrees of freedom. However, a model with more degrees of freedom can generate mosaics that are not physically possible. The mosaic will look great, but the images could be warped in such a way that no camera would have been able to generate it. Therefore, we are also interested in verifying that the generated mosaics are physically possible and computer vision geometry equations hold.

One way to verify physical realism of the generated mosaics is to show that they can be accurately stabilized. WAMI imagery is captured by an aerial platform, such that the ground plane is always visible and is the dominant feature in the image. In this case, computer vision

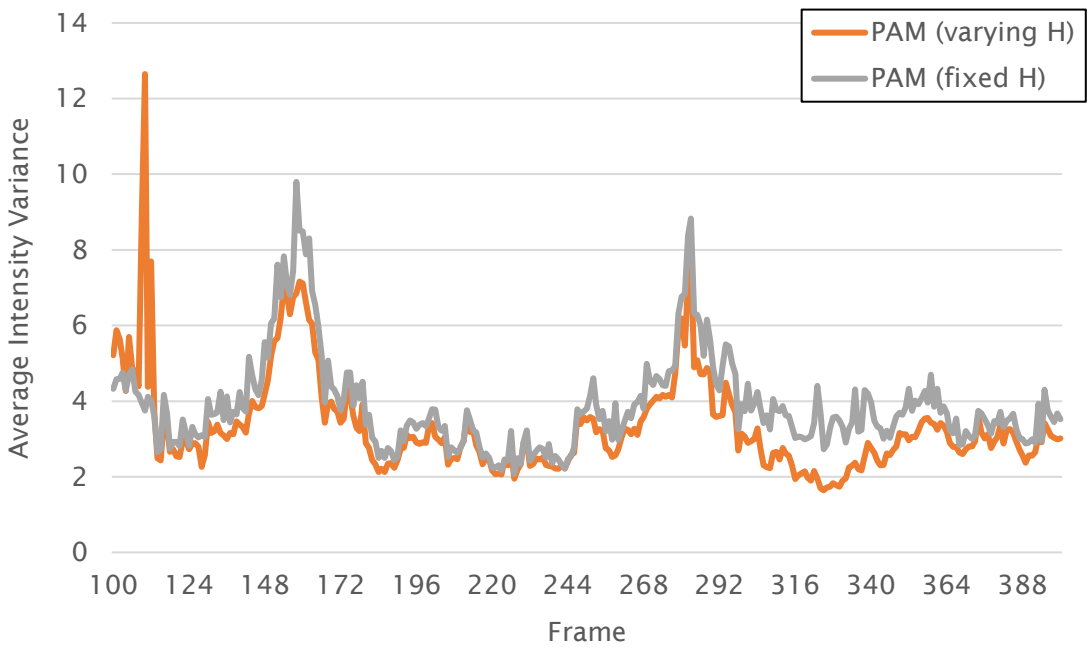


Figure 2.7: Mosaicking accuracy with respect to fixed/varying homographies.

geometry tells us that the imagery can be stabilized by solving for homographies between the first (reference) frame and the rest of the frames (see Chapter 3 for more details). Therefore, if the mosaics are generated in such a way that they could have been captured by a single real camera, the stabilization should be "perfect" and there should be no other motion in the image except parallax and moving objects. If the generated mosaics are not physically possible, the unrealistic deformations in the image would not follow computer vision geometry, and would show up as extraneous motion in the ground plane stabilized imagery. Following this approach, we selected a point on the ground plane and on an edge in the stabilized imagery, and measured its intensity over time. If this point is on the surface of a material with lambertian reflectance, its intensity should be constant over time. If the stabilization is innacurate, and the edge moves across this point, there will be a drastic change in intensity. Figure 2.8 shows the results

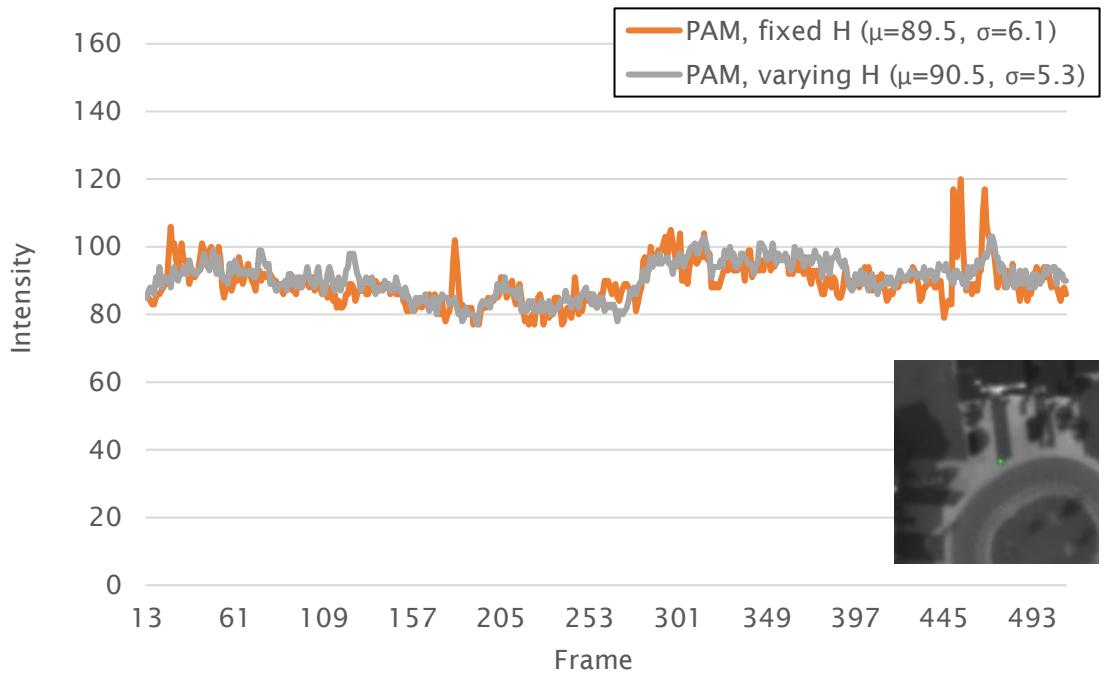


Figure 2.8: Stabilization accuracy with respect to fixed/varying homographies as measured by the image intensity at a point on a plane (indicated by green cross above). The intensity at the selected point should have low variance in an accurate stabilization. Mosaics with fixed homography parameters do not produce as accurate stabilization as mosaics with varying homography parameters.

of this experiment for mosaics with fixed and varying homographies. Mosaics with varying homography parameters do indeed have more uniform intensity over time at the selected point than mosaics with fixed homographies. This is an indication that there is indeed some camera movement in the sensor array, though it appears to be slight.

Qualitative results of the piecewise affine model are illustrated in Figure 2.10. The piecewise affine model has practically perfect registration. We also evaluated the proposed intensity correction algorithm qualitatively. As Figure 2.9 shows, our approach removes most of the intensity differences between the different cameras. Any remaining differences will disappear after blending the images [12].

(a) Initialization (b) After correction

Figure 2.9: Results of intensity correction on frame 283 of the Wright-Patterson dataset. Variations in intensity disappear when the proposed approach is applied.

Figure 2.10: Detailed results of an estimated mosaic on frames 199 and 289. The arrows indicate the endpoints of the seam. The alignment is perfect.

2.4 Conclusions

Mosaicking a sensor array from WAMI is the first step in the semantics extraction process. This step is critical, because all the following vision tasks depend on it being reliable. We have proposed an approach that generates high-quality mosaics in an efficient manner, suitable for such tasks. A comparison with the standard radial distortion model used in previous work reveals that the proposed model is superior.

CHAPTER 3

Video Stabilization

By performing mosaicking as described in the last chapter, we now have a single video stream to analyze and infer useful information from. As mentioned earlier, this video stream covers a large geographic area on the order of a few square kilometers, yet it has high enough resolution that vehicles (and people for future sensors) are visible. People's daily activities and movements contain a lot of useful information, and understanding them is critical to support the applications outlined earlier.

Detecting motion in the video stream is a necessary first step in achieving this understanding. Motion in a video manifests itself as changes in image intensity. Therefore, a simple way to detect motion is to subtract two images, and look for areas in the resulting image where there is a difference. This works (ignoring lighting changes in the scene and CMOS sensor noise), but it does not distinguish between motion caused by objects in the scene and camera motion. In our case, the camera is mounted on an airplane, so camera motion is significant

and can not be ignored. Since we are only interested in the motion of objects, this motion must be removed. Removing (or constraining) the motion of a camera in a video is called video stabilization.

3.1 Related Work

The ideal way to do video stabilization is to recover the 3D scene structure and camera pose (rotation, translation) in every frame, and reproject the video into a static camera pose (or a sequence of synthetic camera poses in some applications). When the (static) 3D world is represented as a point cloud $\{X_i|i = 1 \cdots N\}$, in the coordinate system of camera c as $\{X_i^c|i = 1 \cdots N\}$, its projection in view c as $\{x_i^c|i = 1 \cdots N\}$, camera poses as $\{(R_t, T_t)|t = 1 \cdots T\}$, and images as $\{I_t|t = 1 \cdots T\}$, a stabilized image sequence $\{I_t^s|t = 1 \cdots T\}$ can be generated using

$$I_t^s(x_i^s) = I_t(x_i^t) \tag{3.1}$$

$$I_t^s(K(X_i^s)) = I_t(K(X_i^t)) \tag{3.2}$$

$$I_t^s(K(R_s X_i + T_s)) = I_t(K(R_t X_i + T_t)) \tag{3.3}$$

$$I_t^s(K(X_i^s)) = I_t(K(R_{ts} X_i^t + T_{ts})) \tag{3.4}$$

$$I_t^s(x_i^s) = I_t(K(R_{ts} X_i^t + T_{ts})) \tag{3.5}$$

where K is the camera intrinsics matrix [32], and (R_{ts}, T_{ts}) is the rigid transformation from camera t to camera s, which is easily computed from (R_t, T_t) and (R_s, T_s). Even though conceptually straightforward, this approach is rare to find in practice, because 3D reconstruction (obtaining $\{X_i\}$) and camera pose estimation (obtaining (R_t, T_t)) remain challenging problems in computer vision. Nevertheless, there have been attempts to do this [55] with good results.

The only way to avoid full 3D geometry inference is to make assumptions about the 3D world or camera motion. In wide area motion imagery, the common approach is to assume the 3D structure can be well approximated by a plane. In this case,

$$N^T X_i = d \tag{3.6}$$

$$\frac{N^T X_i}{d} = 1 \tag{3.7}$$

where N and d are the plane parameters (in world coordinate system). Representing the plane parameters in view c as (N^c, d^c), reprojection is then

$$
\begin{aligned}
I_t^s(x_i^s) &= I_t(K(R_{ts}X_i^t + T_{ts})) && (3.8)\\
I_t^s(x_i^s) &= I_t(K(R_{ts}X_i^t + T_{ts}((N^{t^T}X_i^t)/d^t))) && (3.9)\\
I_t^s(x_i^s) &= I_t(K(R_{ts}X_i^t + ((T_{ts}N^{t^T})/d^t)X_i^t)) && (3.10)\\
I_t^s(x_i^s) &= I_t(K(R_{ts} + ((T_{ts}N^{t^T})/d^t))X_i^t)) && (3.11)\\
I_t^s(x_i^s) &= I_t(KHX_i^t) && (3.12)\\
I_t^s(x_i^s) &= I_t(KHK^{-1}x_i^t) && (3.13)\\
I_t^s(x_i^s) &= I_t(\hat{H}x_i^t)\ . && (3.14)
\end{aligned}
$$

This shows that by making the assumption that the world is planar, we can perform the reprojection by simply estimating a 2D projective transformation $\hat{H}$ from view t to view s. The 2D transformation H is called a homography and contains the motion of the camera. This is the same homography as in the previous chapter, although there the plane is at infinity ($d = \infty$). The estimation of $\hat{H}$ is much easier than estimating the 3D world and camera poses, because it has only 8 degrees of freedom. This is in contrast to $3N + 6$ degrees of freedom for the general case.

As we briefly discussed in section 2.2.1, there are two general classes of methods for the estimation of parametric motion models, such as a homography: feature-based and direct. In a direct method, given an initial estimate of the motion, the parameters are adjusted such that

the intensity difference at overlapping pixels between the images is minimized [9]. In a feature-based method, image features such as Harris corners [31] or local extrema in a Laplacian image pyramid are first extracted in both images, then represented with an invariant descriptor such as SIFT [57], and matched by minimizing a descriptor difference. The resulting point correspondences are then used in a robust estimator (tolerant to some wrong correspondences), such as RANSAC [24].

The advantage of a direct method is that it usually produces more accurate parameter estimates than a feature-based method, because it uses the entire image in the minimization of the energy function, and does not require the estimation of features and their correspondences, which may be difficult to find and match. The disadvantages of a direct method are that it requires a good initial estimate of the parameters, is usually slower than a feature-based method, and it assumes there is no other motion besides the motion of the camera (all overlapping pixels are considered for optimization). The first problem can be overcome relatively easily, and we could deal with the second problem as in the previous chapter, by carefully selecting a small number of pixels to be used in the optimization. However, the third problem is more difficult to handle. In our imagery, there are hundreds, if not thousands, of moving targets, and their motion would cause problems in the optimization and prevent us from obtaining the correct homography estimate. A feature-based method paired with a robust estimator does not have this problem, because correspondences that do not agree with the homography model are automatically removed from consideration. Therefore, we will use a feature-based method in our approach.

3.2 Approach

We need to estimate homographies from each frame of the video to a reference frame, usually the first frame of the sequence. We can estimate these directly, by finding correspondences between each frame and the first, or indirectly, by finding correspondences between adjacent frames and composing (multiplying) the resulting transforms to compute the desired homography from each frame to the first. Each of these strategies has some advantages and disadvantages. The first strategy is generally more accurate than the first, but because we are estimating each homography independently, a sequence of such homography estimates is not guaranteed to be consistent (the estimate of the plane parameters may slightly vary from frame to frame). This inconsistency manifests itself as jitter in the stabilized sequence, especially for frames that have been taken from different viewpoints than the first, and the number of correspondences is small. The second strategy does not have a problem with jitter, because the homography estimates there are not independent. However, it is generally less accurate due to the accumulation of small errors in parameter estimates over time (drift). Drift and jitter are both undesirable.

We propose an approach that takes both sets of homography estimates, and produces a new set minimizing both drift and jitter. Let the set of homographies estimated directly be $H_F = \{H_{t,1} | t = 2 \cdots T\}$ (from frame t to 1) and the set of pairwise homographies be

$H_P = \{H_{t,t-1} | t = 2 \cdots T\}$ (from frame t to $t-1$). If the homographies are correct, they need to satisfy the following set of constraints:

$$H_{t+1,1} = H_{t,1} H_{t+1,t} \tag{3.15}$$

$$H_{t+1,t} = H_{t,1}^{-1} H_{t+1,1} \tag{3.16}$$

$$H_{t,1} = H_{t+1,1} H_{t+1,t}^{-1} \, . \tag{3.17}$$

Because H_F and H_P were estimated independently, these constraints will not be satisfied in general. Therefore, we devised an iterative optimization procedure to adjust H_F and H_P until they agree. In doing so, we minimize jitter error present in H_F and drift error present in H_P. In each time step $t > 2$, we repeatedly do the following:

$$H'_{t+1,1} = H_{t,1} H_{t+1,t} \tag{3.18}$$

$$H_{t+1,1} = 0.5 \times H_{t+1,1} + 0.5 \times H'_{t+1,1} \tag{3.19}$$

$$H'_{t+1,t} = H_{t,1}^{-1} H_{t+1,1} \tag{3.20}$$

$$H_{t+1,t} = 0.5 \times H_{t+1,t} + 0.5 \times H'_{t+1,t} \tag{3.21}$$

$$H'_{t,1} = H_{t+1,1} H_{t+1,t}^{-1} \tag{3.22}$$

$$H_{t,1} = 0.5 \times H_{t,1} + 0.5 \times H'_{t,1} \, . \tag{3.23}$$

The difference between $H'_{t+1,1}$ and $H_{t+1,1}$, $H'_{t+1,t}$ and $H_{t+1,t}$, $H'_{t,1}$ and $H_{t,1}$ does not go to 0 immediately, but it decreases with every update. Therefore, we repeat the above sequence within each time step as many times as necessary until the differences are negligible.

The order in which the homographies are updated matters and is carefully designed. We first modify $H_{t+1,1}$ in which we have the least confidence, since $H_{t,1}$ was optimized in the last time step and $H_{t+1,t}$ is a homography for adjacent frames, which has small error. We then update $H_{t+1,t}$, which is now the remaining unoptimized homography. Finally, $H_{t,1}$ is updated using the new values of $H_{t+1,1}$ and $H_{t+1,t}$.

3.3 Results

The proposed stabilization method was evaluated on a sequence from the Wright-Patterson dataset [51], which is a real WAMI dataset captured by the Air Force Research Lab. The dataset was mosaicked using the method proposed in the last chapter before performing stabilization. Ground truth stabilization was not available, as this would require manually estimating correspondences for every frame in the dataset. Therefore, we devised an alternative approach for quantitative evaluation.

If the dataset is correctly stabilized, the image intensity for locations on the ground plane and never occluded by nearby trees or structures should be (nearly) constant. This assumes the material on the ground has Lambertian reflectance. There may be small variation in intensity due to the changing relative position of the sun (light source) with respect to the camera, because the captured image is brighter when the sun points into the camera. An additional

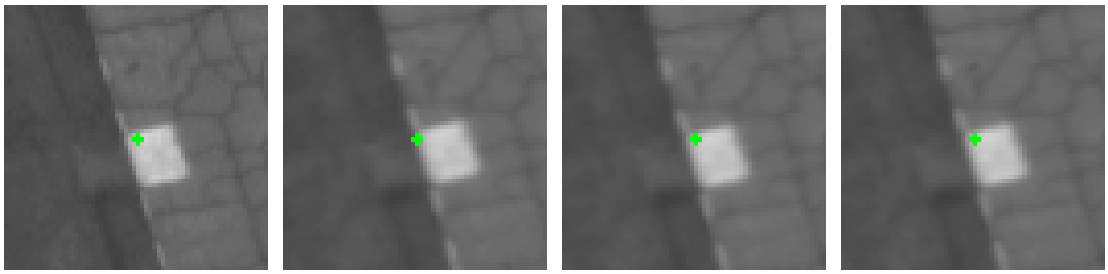
source of intensity variation may be the automatic gain and exposure control in the sensor, but most of this variation is removed using the method in section 2.2.2 adapted to a sequence of images (rather than an array), which we apply before evaluation.

Therefore, we selected a random image coordinate with the above property and measured its image intensity in every frame of the stabilized sequence for the different stabilization methods. We chose a point with a high image intensity gradient (edge), so that small errors in stabilization would be immediately apparent. The local image region around this point is shown in Figure 3.1. We also show the same region in the last frame of the stabilized sequence for different stabilization methods.

The results are shown in Figure 3.2. When pairwise homographies are used for stabilization, drift in parameter estimates is clearly evident. The image intensity varies significantly, indicating image motion. When direct homographies are estimated, image intensity varies little in general, which is already an improvement, but the stabilization is not "smooth." There are many frames where the image moves back and forth (sharp changes in intensity from frame to frame). This is evidence of the jitter phenomenon. Finally, when homographies are optimized using the proposed approach, we see that both the image intensity varies very little and the amount of jitter is significantly reduced.

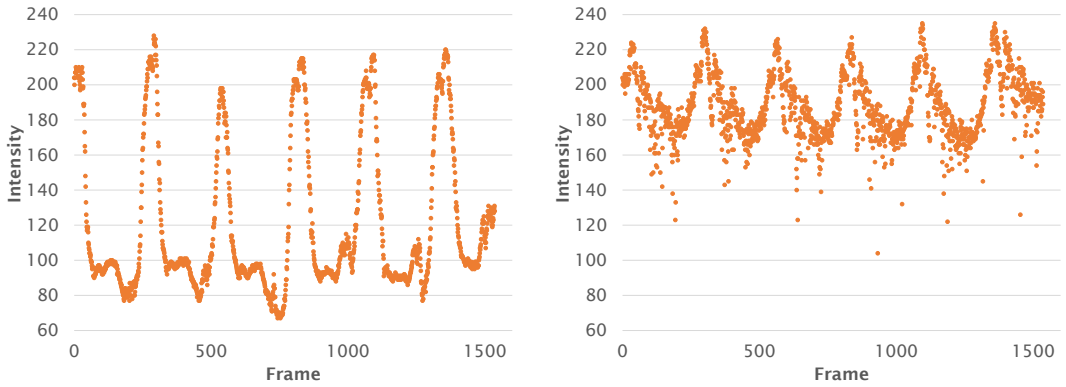
3.4 Conclusions

Accurate image stabilization is important for the detection of motion, enabling tracking of targets of interest. The ideal way to stabilize the video is to perform full 3D geometry inference,



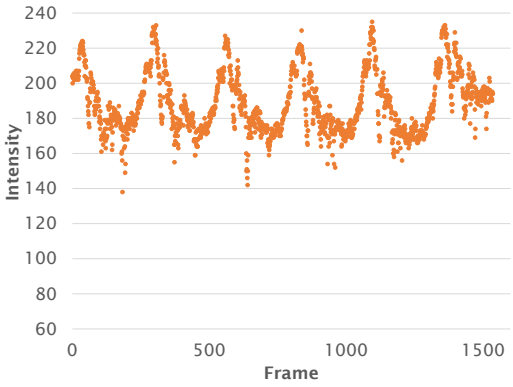
(a) (b) (c) (d)

Figure 3.1: Evaluating stabilization accuracy. (a) The image region in the first frame of the sequence, with the evaluation coordinate marked. The image region in the last frame of the sequence when stabilized with (b) pairwise homographies, (c) direct homographies, and (d) optimal (proposed) homographies.



(a) Using pairwise homographies

(b) Using direct homography estimates



(c) Using optimally estimated homographies

Figure 3.2: Stabilization accuracy for different estimation methods as measured by the intensity at a fixed image coordinate over time.

but this remains an impractical approach. Instead, we stabilize the video by assuming the world is planar and estimating homographies, a common technique for aerial videos. Our proposed homography estimation algorithm has very little drift and jitter, in contrast to widely used existing techniques.

CHAPTER 4

Tracking

The problem of tracking multiple targets in a video sequence has appeared in a wide variety of contexts, including radar tracking in the early days, pedestrian tracking, cell and ant tracking in biology, and vehicle tracking. In all cases, the goal is to determine a spatio-temporal description of the moving objects in the sequence. As is often the case in computer vision, each of these contexts presents a different set of challenges and corresponding solutions. In this chapter, we revisit the multiple target tracking problem in the context of wide area motion imagery (WAMI) imagery.

As in other types of imagery, one needs to handle occlusions, and changes in illumination and appearance. Unlike other imagery, the large, unknown, and varying number of targets along with a large number of false detections makes this a particularly difficult data association problem. Furthermore, the mapping between detections and targets is many-many in every

frame in general, which increases the complexity of the computation. A correct data association requires looking ahead in time, but this in turn causes an exponential growth of the search space.

A classical formulation of the multiple target tracking problem over $N > 1$ frames is a multidimensional assignment problem [66]. Here, the objective is to partition all the detections into tracks and false alarms, such that each detection is assigned to at most one track, and the posterior probability of each track is maximized. This problem is NP-hard, and is usually approximately solved using Multiple Hypothesis Tracking (MHT) [69, 17]. This method operates on only a subset of data association hypotheses, but the subset is chosen and maintained optimally. Nevertheless, in our domain of WAMI imagery, this becomes costly even for small N, because the number of targets and potential interactions between them is large. A large number of targets increases the size of the linear assignment problems to be solved, and a large number of potential interactions increases the number of candidate hypotheses to be maintained and generated.

Even when $N = 2$, and the multidimensional assignment problem is reduced to a linear (1D) assignment problem, the computation is still intractable. This was observed by Reilly et al [70], where they proposed to divide the image into cells and solve the association problem independently within each cell. Unfortunately, with no frames to look ahead, the ambiguity in association is large, and local scene constraints need to be imposed to get correct association. Dividing the image into tiles was also proposed in [45] where an MHT tracker with $N = 3$ was used.

In our work, we argue that using a large sliding window (value of N) is essential in minimizing false alarms and achieving high tracking accuracy in WAMI imagery. However, in order to resolve the huge number of possible associations at such scale, it is not possible to use existing approaches and a new way of looking at the problem is needed. We propose a new approach which, with a few relaxing assumptions, is able to significantly reduce the search space of possible associations, and can solve the association problem very efficiently. It does so purely by maximizing motion smoothness and appearance consistency. No additional scene constraints are needed.

4.1 Related Work

Another class of methods that solves the multiple target tracking problem over multiple frames is based on min-cost flow/Linear Programming [41, 93, 3, 64, 8]. The advantage here is that a globally optimal solution can be obtained, sometimes very efficiently [64, 8]. However, these methods have a few inherent assumptions unsuitable in our domain. First of all, they can only express "static" association costs. They make it difficult to model long-range, dynamic, motion and dependencies. What we mean by this is that the association cost for a pair of detections in these formulations is independent of all other associations that happened earlier in a track. For example, if at time t a target is traveling at a speed s units/frame (which we have determined with associations up to time t), we would like to express that at time $t+1$, the association with a detection s units away is more likely than with a detection $s-\Delta s$ or $s+\Delta s$ units away. This "constant velocity" motion model cannot be expressed with variables over

pairs of detections. This can be alleviated by introducing more variables (e.g. over triplets), but this comes at a high cost. Workarounds have been proposed [3], but in general, these methods have weak motion models and often require a prior specification of locations where objects enter the scene and exit. In our domain appearance modeling is limited, and motion modeling is important. Furthermore, as a practical matter, the association cost in existing implementations is usually expressed in terms of the overlap of two detections, which will not work in our domain where the frame rate is low (there is no overlap).

Due to these characteristics, these methods are less tolerant to false detections, may create many short tracks and require pre or post-processing [39]. Hierarchical and tracklet-based methods present another alternative that has become popular [63, 2, 34, 40]. They first determine short tracks, or tracklets, and then link these tracklets into longer tracks in one or more steps. Tracklets are usually determined by using a nearest neighbor association [63], some affinity measure [34], or particle filtering [40]. They are then combined by solving the linear assignment problem or using min-cost flow [73, 13]. In most, if not all cases, these tracklets are initially very short, covering as few as 3 frames. This puts more pressure on the linking step to correctly combine the tracklets. In contrast, we argue in our work that it is useful to solve for longer tracklets covering many frames, and then use a simpler linking step. This approach was avoided in the past, because it was thought to be cost prohibitive. We show that it is possible to resolve the great number of association hypotheses at long time scales and efficiently generate long tracklets.

Particle filtering has usually been a classic solution for single-target tracking, but there have been several attempts to use it for multiple target tracking as well [46, 62]. The difficulty there is to keep the number of (joint) state hypotheses low, while making sure the correct state is not missed. In [46], a more efficient solution is obtained by assuming Gaussian motion, and computing the target state analytically for each sampled data association. Another sampling algorithm, using data-driven MCMC, was introduced in [92]. Spatio-temporal smoothness in motion and appearance was key to recovering the tracks of an unknown number of targets. This approach, however, requires a good initialization for quick convergence.

4.2 Approach

The goal of our algorithm is to infer tracklets, each representing one object, over a (sliding) window of frames. This window is usually 4-8 seconds in length. The input to our algorithm is a set of object detections (blobs) in each frame. These can be as simple as connected components taken directly from background subtraction, or they can be the output of a more complex object detector. Each object detection also has an associated appearance representation, such as the raw image patch, or a histogram. These tracklets are then aggregated into tracks by continuously performing track linking. Since the tracklets we determine are long, there is little ambiguity in track linking, and this step is relatively simple. The key is how to efficiently determine the tracklets over a long temporal window. A flow-chart that illustrates multi-object tracking using tracklets is in Figure 4.1.

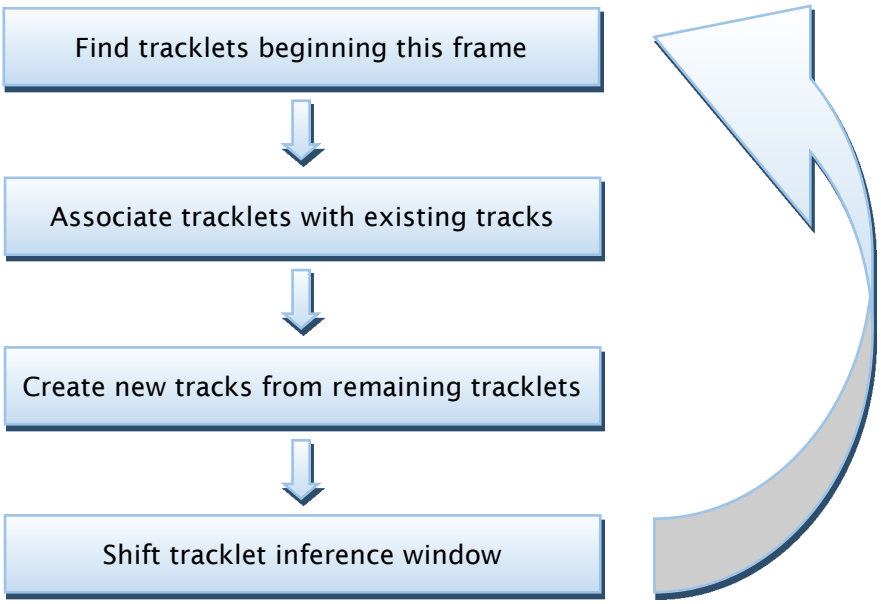


Figure 4.1: The role of tracklets in multi-object tracking. The first step is the focus of this chapter.

4.2.1 Detection

Since the resolution of targets in our imagery is limited, we rely on background subtraction for the detection of moving targets. Naturally, not all detections from background subtraction correspond to targets of interest. To minimize the number of detections we need to consider without attempting to model the target appearance, which is difficult given the limited resolution, we train a binary decision tree on object area and aspect ratio features.

Given a few frames from a training sequence with all target objects marked with a bounding box, and a structure of the decision tree, we need to learn the thresholds at each level. There are 3 levels in our decision tree. We first check the aspect ratio is less than some threshold, then the object area greater than some threshold, and finally the object area less than some

threshold. At each level, the threshold is determined by maximizing a weighted F-score, which allows us to put more emphasis either on precision or recall.

4.2.2 Tracking

We do not assume an a priori number of targets in the scene, and the number can vary over time. Requiring that each detection be used at most once in a track, and running MHT over an 8 second (16 frame) window would be cost prohibitive. However, we believe that tracklets determined over such long time scales are useful. Therefore, instead of reducing the size of the temporal window, as has been done by others, we instead relax the assumption that each detection be used at most once. This allows us to determine tracklets independently, which simplifies the computation and allows for parallelism.

Since we do not know the number of targets ahead of time and we would like to avoid computations jointly involving all targets and detections, we assume that each detection in the first frame of the window is a potential object. Therefore, we find an optimal tracklet, or a set of tracklets, starting at each detection in the first window frame. This is not a problem, because for detections that are false alarms, the model of a valid tracklet (consistency of motion and appearance) is not satisfied, and the tracklet is discarded. Tracklets that start in the second or later frame of the window are found when the sliding window shifts to that frame.

4.2.3 Problem formulation

If the initial detection of an object is given to us, we know there must be another detected instance of that object located "nearby" in subsequent frames. We are assuming there are no missed detections (due to occlusion or else) for now. Therefore, the optimal tracklet, or a set of tracklets, that we want to find must be composed of a series of "nearby" detections. This can be expressed in a directed acyclic graph, which we call a detection tree, or an association tree. For a window size of T frames, this tree would have T levels. A node in level t has links to those nodes in level $t + 1$, which are "nearby." The root of the tree, $t = 0$, represents the initial detection. The definition of nearby for the first level is set according to the maximum expected velocity of the target, but for the following levels, it is determined dynamically by maximum possible acceleration from the target's current velocity. The velocity estimate is maintained using a Kalman filter for each sequence of detections (each leaf in the tree). This graph construction is actually an important point. As mentioned above, min-cost flow/Linear Programming methods have no notion of a "track" during graph construction and create more edges than necessary and set the costs on those edges to reflect only a weak motion model.

The number of possible tracklets arising from this detection tree is huge (greater than or equal to the number of paths), and we certainly do not want to evaluate every hypothesis. Instead, we realize that every such hypothesis is making a decision about including or not including each detection. In other words, this is just a binary labeling, or segmentation problem. The valid detections need to be separated from the invalid detections. The valid detections are those that have similar appearance to the initial detection and have smooth motion. The

invalid detections are detections due to noise, or due to targets other than the one that generated the initial detection. One consequence of this view is that given the valid detections, it is not always known which targets generated them. It could be that a single target generated them (with possibly multiple detections in one frame due to the noisy nature of background subtraction), or it could be two (or several) similar looking targets very close to each other. It may seem that nothing was gained by the segmentation, but actually solving this problem is easier than before, because the search space is significantly reduced.

There are several ways to solve the segmentation problem. One way is to use a min-cut formulation similar to [29]. This, however, produces a "hard" segmentation without a confidence estimate, and restricts the form of the interaction between different detections. An alternative way that we pursue here is to determine the labeling in a generic probabilistic framework.

4.2.4 Segmenting Detections

Let the label of each detection i at frame t be a binary random variable y_i^t, and let $\mathbf{y}$ denote all labels in the window. Let the observed properties of each detection (location, appearance, etc.) be denoted as $\mathbf{o}_i^t$, and let $\mathbf{o}$ denote all observations Then, the segmentation problem is

$$\arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{o}) \ . \tag{4.1}$$

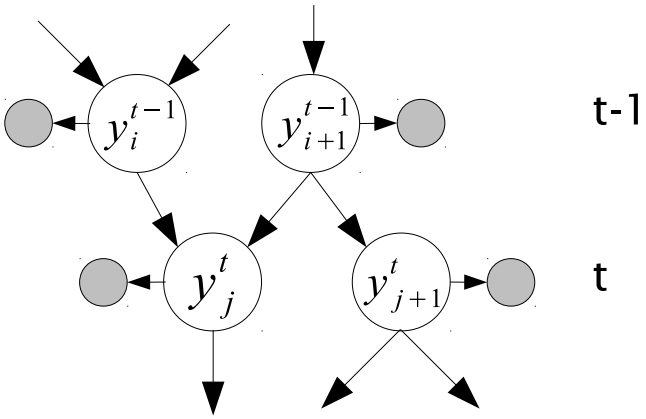


Figure 4.2: Example of the structure of the graphical model. Each y_i^t is a binary variable that represents a detection label. The shaded nodes represent the measurements associated with each detection (location, appearance, etc.).

Solving this problem depends on how the joint distribution is formalized, allowing a great deal of flexibility. Here we factorize the joint distribution into a product of prior and conditional probabilities,

$$p(\mathbf{y}, \mathbf{o}) = p(y^0) \prod_{\substack{i,j,t>0 \\ y_i^t \text{near } y_j^{t-1}}} p(y_i^t | y_j^{t-1}) \prod_{i,t>0} p(\mathbf{o}_i^t | y_i^t) \; . \tag{4.2}$$

We let $y^0 = 1$ to denote the assumption that each detection in the first frame is valid. $\mathbf{o}_i^t$ denotes the observed properties of each detection (location, appearance, etc.). This factorization corresponds directly to the detection tree discussed earlier, and is illustrated in Figure 4.2.

One assumption that we are making here that may not be immediately apparent is that when a detection has multiple parent detections, independence of the parents is assumed. As a result, the conditional distribution conditioned on multiple parents is factorized into a product of simple conditional distributions, each conditioned on only one parent:

$$p(y_i^t | y_1^{t-1}, y_2^{t-1}, \cdots, y_K^{t-1}) \propto \prod_{k=1}^{K} p(y_i^t | y_k^{t-1}) \; . \tag{4.3}$$

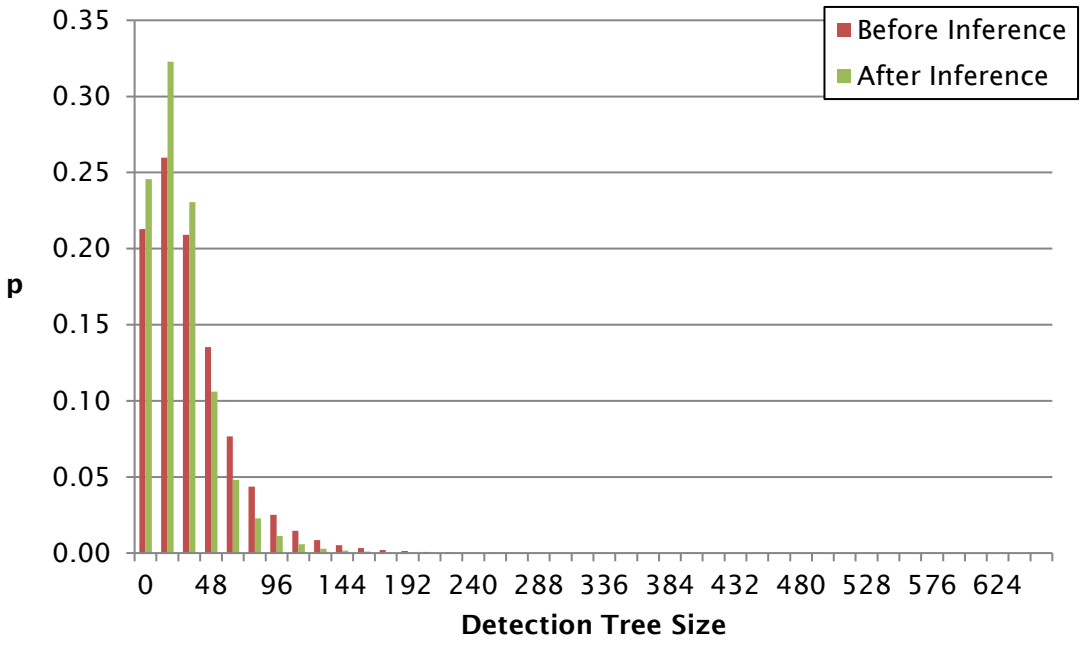


Figure 4.3: The effect of performing a detection segmentation step. The size of the detection tree, and thus the hypothesis space, is significantly reduced.

This assumption is not correct when the parent detections come from one object (i.e. split detections caused by noisy background subtraction), but it significantly simplifies the computation, and is not a big problem in practice. Without this assumption, the conditional probability table for a distribution conditioned on K parents would be 2 by 2^K.

Once the prior and conditional probabilities are specified, the optimal solution to the segmentation problem is given by MAP inference. In addition, the max-marginals provide us with a confidence estimate of each detection. MAP inference is a well-studied problem, and can be solved using the max-product algorithm [49] as done here, or LP relaxation algorithms, such as [81]. The effect of this segmentation step can be seen in Figure 4.3, where we show how the search space is reduced after invalid detections are removed from the graph.

	y_i^t = 0	y_i^t = 1
$p(\mathbf{o}_i^t \mid y_i^t)$	$1 - a(\mathbf{o}_i^t, \mathbf{o}^0)$	$a(\mathbf{o}_i^t, \mathbf{o}^0)$

Table 4.1: Form of the conditional probability distribution $p(\mathbf{o}_i^t|y_i^t)$ used in the graphical model.

	y_i^t = 0	y_i^t = 1
$y_j^{t-1} = 0$	0.5	0.5
$y_j^{t-1} = 1$	$1 - a(\mathbf{o}_i^t, \mathbf{o}_j^{t-1})m(\mathbf{o}_i^t)$	$a(\mathbf{o}_i^t, \mathbf{o}_j^{t-1})m(\mathbf{o}_i^t)$

Table 4.2: Form of the conditional probability distribution $p(y_i^t|y_j^{t-1})$ used in the graphical model.

4.2.5 Probability Distributions

The conditional probability $p(\mathbf{o}_i^t|y_i^t)$ reflects the appearance similarity between the corresponding detection y_i^t and the initial detection y^0. Any appearance similarity measure can be used. It can be as simple as a sum of squared differences, or as complex as output of a classifier. For appearance similarity a that ranges in $[0, 1]$, the distribution is shown in Table 4.1.

The conditional probability $p(y_i^t|y_j^{t-1})$ is based on both the appearance similarity between the corresponding detections, as well as the motion likelihood of this detection given the preceding detections. The preceding detections are those which are on the path up to the root in the detection tree. There is a problem with this definition when a particular detection has multiple parents, because the motion model, which is described below, assumes only one observation at each timestep. To solve this problem we take the parent detection which gives the maximum motion likelihood, and call it the "motion parent." The effect of this is not to unfairly penalize valid detections that follow this ambiguity. For a motion likelihood m that ranges in $[0, 1]$, the conditional probability table is shown in Table 4.2.

This conditional distribution is a little bit complicated due to the asymmetry. The asymmetry is necessary, because we need a different behavior when the parent label is 0 (invalid detection) and when it is 1 (valid detection). When the parent detection is valid (bottom row), the distribution expresses that the probability is high when the detection label appears similar to the parent, and the motion likelihood after observing this detection is high (the motion is smooth). When the parent detection is invalid, however, we can not say anything about the appearance similarity. This is because a parent detection that is a false alarm (noise) may still have a similar appearance and motion. Therefore, in this case the distribution does not give a "hint" about the label of the detection.

Any motion model can be used to determine the motion likelihood. Here we use a simple linear-Gaussian model:

$$\mathbf{z}_{t+1} = \mathbf{A}\mathbf{z}_t + \mathbf{w} \tag{4.4}$$

$$\mathbf{x}_{t+1} = \mathbf{H}\mathbf{z}_{t+1} + \mathbf{v} \tag{4.5}$$

where $\mathbf{z}_{t+1}$ is the state vector, which includes the object position and velocity, $\mathbf{x}_{t+1}$ is the measurement vector of the object position, $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the process noise, and $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the measurement noise. The motion likelihood m is then

$$m(y_i^t) = \exp\left(-\frac{1}{2}\mathbf{e}_i^T \mathbf{P}_i^{-1} \mathbf{e}_i\right) \tag{4.6}$$

where $\mathbf{e}_i = \mathbf{z}_t^i - \hat{\mathbf{z}}_t^i$ and $\mathbf{P}_i = \mathbf{A}\hat{\mathbf{P}}_i\mathbf{A}^T$. $\hat{\mathbf{z}}_t^i$ is the posterior state estimate, $\mathbf{P}_i$ is the prior state covariance, and $\hat{\mathbf{P}}_i$ is the posterior state covariance in the previous time step. The model is initialized using the first two detections in the sequence.

Having specified the joint distribution, the MAP labeling is given by MAP inference. As indicated earlier, the valid detections in this labeling do not always define one tracklet. There may be extra detections due to noise, or due to the limitations of our model (a simplified factorization of the joint). We describe how to handle this problem next.

4.2.6 Tracklets from detections

Since we made the assumption that the first detection in our graph comes from a potential target, we continue this assumption, and look for a sequence of detections from the root that are labeled as "valid." There may be more than one sequence, but we choose the one(s) that has the maximum length. We find this sequence by performing Breadth-First-Search (BFS). An alternative way to generate possible tracklets is to follow the motion parent pointers up to the root of the tree from each valid detection, and removing any tracklet that is a prefix of another. We have considered this approach, but performing BFS is much faster, and this is our method of choice.

At this point we have a set of tracklets, but not all of them have been generated by real targets. Some of them are due to parallax, reflections, or other noise. To remove false tracklets, we analyze tracklets' features, and remove those tracklets that fail to pass our decision criteria.

Currently we are using a very simple, rule-based, classifier, and it works well, but a more powerful, or application-specific, statistical approach would work just as well.

Our criteria for a valid tracklet are:

- Minimum Length: The number of detections in the temporal window must be at least 75% of the window size.

- Acceleration: The average acceleration of the object must be less than a threshold (≈ 6 m/s^2). The average acceleration is estimated from the position of the detections, which are at least half a second apart. The elapsed time requirement is necessary to avoid bad estimates due to discretization noise.

- Smooth Motion: To determine motion smoothness, we compute the dot product of successive motion directions (at least half a second apart), transform it to $[0, 1]$ range, and compute the average. This average must be greater than a threshold (≈ 0.80) for the criterion to be satisfied. Note that this does not rule out tracklets that are making a turn, since the large change in direction will be filtered out by the average.

- Minimum Speed: The target moves with minimum speed. The effect of this is to filter out motion due to parallax. This only applies when detections are obtained from background subtraction.

The importance of each criterion is illustrated in Figure 4.4. The minimum length criterion is responsible for removing most of the tracklets.

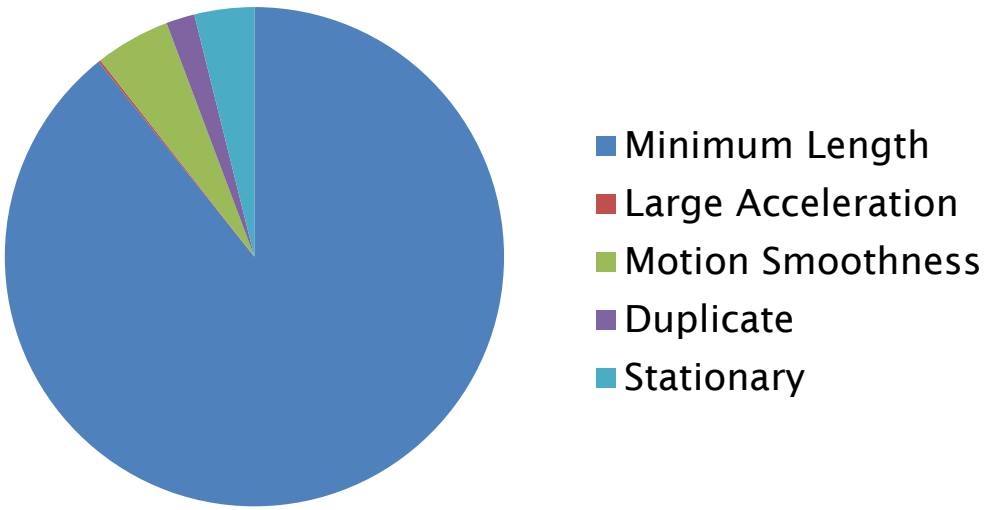


Figure 4.4: Criteria for validating found tracklets. The most important criterion for track validation, as measured by the fraction of tracklets removed by it, is the tracklet length.

Background subtraction may generate multiple detections for one target. This would mean the generation of two or more tracklets that are virtually identical (differing in only 1 or 2 frames). To handle this case, we also incorporate a duplicate detector, which detects and removes duplicate tracklets. It computes the similarity of all pairs of tracks τ_1 and τ_2 using

$$sim(\tau_1, \tau_2) = \frac{1}{T} \sum_{1}^{T} a(\tau_1^t, \tau_2^t) * s(\mathbf{b}_1^t, \mathbf{b}_2^t) \tag{4.7}$$

$$s(\mathbf{b}_1, \mathbf{b}_2) = \exp(-\|\mathbf{b}_1 - \mathbf{b}_2\|/c) \ , \tag{4.8}$$

where $a(\cdot)$ is the appearance similarity measure as before, $\mathbf{b}$ is the bounding box of the detection (x, y, w, h), and c is a constant to increase the dynamic range ($c \approx 40$). When $sim(\tau_1, \tau_2)$ exceeds a threshold (≈ 0.55), the track with the lower motion smoothness is removed.

4.2.7 Occlusion handling

So far in the discussion we have assumed there is no occlusion or missing data. It turns out that when an object is occluded but the occluder is detected, the algorithm as presented still works. This is because the detection tree does not really change, except that no detections in the frame where the object is occluded will be valid. A tracklet is still found, provided that the object is not occluded for most of the window. In that case, the tracklet would fail the first criterion above.

The real problem that needs to be handled is missing detections. When there is a missed detection, the detection tree will be shorter than T. If it is too short, any tracklets that are found will not have enough detections and fail the first criterion above. This problem is solved by adding "virtual detections" to the detection tree. These are added whenever a detection in frame t has no nearby detections in frame $t + 1$. The position of this virtual detection is estimated using the motion model, and the appearance is copied from the (detected) parent. This procedure is recursive, so that when a newly added virtual detection does not have nearby detections in the next frame, the process is repeated.

4.2.8 Track Linking

A multi-target tracker based on tracklets found by the presented algorithm works as shown in Figure 4.1. The step that we now address is the second one -- associating tracklets with existing tracks. The task of this step is to form long tracks from tracklets found in the sliding window. Many strategies can be used here (e.g. Hungarian algorithm), but since the tracklets

we find are relatively long, the ambiguity in matching them to tracks is reduced. Therefore, we can use a much simpler strategy, that also allows us to accomodate the many-many mapping property.

We associate each tracklet with an existing track as long as (4.7) is above a threshold. If we are only interested in matching each tracklet to one existing track, then we associate with a track having the maximum similarity, provided the similarity is above a threshold. A reasonable threshold is again ≈ 0.55.

4.3 Results

We have evaluated the multi-target tracker on a publicly available dataset from Air Force Research Laboratory [51]. This dataset was captured by an array of six cameras at roughly 1 Hz, and it is in grayscale. We mosaicked the dataset using [68], stabilized it, and georeferenced it prior to tracking. A reference with a resolution of 0.30 meters (1 foot) per pixel was used, making vehicles, our targets of interest, about 20x10 pixels in size. A 1024-frame sequence of a 1408x1408 region was selected for evaluation. One frame from this sequence is shown in Figure 4.5.

The only moving objects in the video are vehicles, but they are in very low resolution. The small vehicle size makes detection and tracking quite challenging. Since this low resolution gives very limited appearance information, we use a relatively simple approach for measuring

Figure 4.5: One frame of the sequence used in the evaluation. Vehicles, our targets of interest, are quite small.

appearance similarity. We build an intensity histogram for each patch and use the exponentiated negative KL-divergence between the histograms as a similarity measure. This basically allows us to distinguish light-colored vehicles from dark-colored vehicles.

Moving object detection was done using background subtraction. The background is modeled as the mode of a sliding window of 11 frames. A window size of 8 frames, corresponding to about 8 seconds of video was used.

The dataset includes ground truth, which was manually generated, although we made one change to it. In each track, we removed ground truth detections which occured before the vehicle first moved. This means that tracks with vehicles that never move in the ground truth are removed, and other tracks are truncated in the beginning, if needed. We rely on background subtraction for detection, so these detections would never be found and tracked by our algorithm. Furthermore, until a vehicle moves, it is not of interest. We did not remove detections that occurred after a moving vehicle stopped. These would also be missed by our

detection algorithm, but a stopping vehicle case is of interest and we believe that such case should be handled in the future. The edited ground truth contains 403 tracks.

Several metrics were used to measure performance: object detection rate (ODR), false alarm rate (FAR), mean cumulative swaps of tracks (SWPS), and mean cumulative broken tracks (BRKS). Object detection rate is defined as the fraction of detections in the ground truth found in the estimated tracks. False alarm rate is defined as the average number of false detections in estimated tracks in every frame. Mean cumulative swaps of tracks is defined as the average number of swaps (ID switches) in every ground truth track (over its lifetime). Mean cumulative broken tracks is defined as the average number of breaks in every ground truth track (over its lifetime). A break happens when a ground truth track is not matched to any ID in the next frame. These last two definitions are based on [71]. Computational efficiency was measured by the average number of frames processed per second (FPS). We ran the tracker 3 times on an AMD FX-6300 CPU, with "warm" cache (minimizing I/O time), and averaged the runtimes.

We compared our approach with [70]. Our implementation of [70] was configured to use one grid cell and to not perform gradient suppression in order to have the tracker operate on the same set of detections as our proposed algorithm. However, the increase in the number of false detections was compensated by removing tracks shorter than 3 frames. We believe this configuration is the most suitable one for comparison. We ran the implementation without allowing any occlusion ($p = 0$) as well as with allowing 1 occluded frame ($p = 1$). The comparison is shown in Table 4.3.

	Proposed	[70] ($p = 0$)	[70] ($p = 1$)
Object Detection Rate	0.36	0.36	0.39
False Alarm Rate	1.03	53.0	81.5
Track Swaps	0.48	1.33	0.96
Track Breaks	0.64	1.22	0.93
Frames Per Second	42.0	8.73	6.78

Table 4.3: Comparison of the proposed approach with competing methods on real data.

	Proposed	[70] ($p = 0$)
Object Detection Rate	0.91	0.90
Track Swaps	1.22	3.34
Track Breaks	0.44	0.34

Table 4.4: Comparison of the proposed approach with competing methods on ground truth detections.

We also evaluated the data association performance of the proposed algorithm by running it on the set of ground truth detections. In other words, instead of using detections from background subtraction, we generated detections from the ground truth. However, since the ground truth only contains the center of each target, even the little appearance information we were able to take advantage of is now significantly limited. In this experiment, we are effectively tasked with estimating tracks from moving dots. The minimum speed filter and smooth motion filter from section 4.2.6 were disabled for this experiment, but otherwise the algorithm was unchanged. The results of this evaluation are shown in Table 4.4.

The primary parameter of the proposed approach is the size of the sliding window, N. To understand the algorithm's performance with respect to this parameter, we have run the tracker with different values of N on the same evaluation sequence. These results are shown in Table 4.5.

N	ODR	FAR	SWPS	BRKS	FPS
4	0.40	8.17	0.66	0.75	44.3
8	0.36	1.03	0.48	0.64	42.0
12	0.34	0.89	0.34	0.54	40.3
16	0.31	0.73	0.22	0.43	38.4

Table 4.5: Effect of sliding window size on tracking performance.

It is clear from the quantitative evaluation that the algorithm presented is very good at making detection associations. This is supported by the low number of id switches and track breaks on real data in Table 4.3 as well as on synthetic data in Table 4.4. When compared to [70] on real data, the number of association errors is approximately cut in half. On synthetic data, the number of swaps is higher than on real data, likely due to the lack of target appearance information. The number of swaps is high for [70], because the algorithm had trouble keeping track of the same ID on stationary targets. The number of swaps was low for moving targets, but when the metric is averaged for all targets, it becomes very high. Track break performance on synthetic data is worse than [70], but not by much, and it is still quite low.

The object detection rate in real data is low, but this is primarily due to vehicles that stop, are not detected by background subtraction, or move slowly. When vehicles stop, there are no detections from background subtraction available, and the algorithm is unable to track the vehicle any further. Handling this case is the subject of our future work. On synthetic data, the detection rate is nearly perfect. The false alarm metric is not applicable here.

The sliding window size is correlated with all tracking metrics. A longer window size generate more accurate tracking results, at the cost of a decrease in object detection rate. However, even a small window size of 4 frames significantly improves tracking accuracy compared to a

Hungarian algorithm based tracker [70]. These results show that using a multiple-frame inference procedure is essential in achieving accurate tracking results in wide area aerial surveillance imagery, where target appearance information is limited. Thanks to our fast inference procedure, this multiple-frame inference can be very efficient, and run in real-time.

We implemented the algorithm just presented in C++. The implementation is available on the author's website. The runtime of the algorithm depends on the number of detections in the scene. When the number of detections per frame is small (≈ 100), the runtime is around 42 frames per second on a 1408x1408 video.

4.4 Conclusions

Wide Area Motion imagery presents additional challenges for tracking algorithms due to its low sampling rate, limited resolution of targets to track, and a high number of targets. We presented an algorithm that solves such large data association problems by using a long sliding window and by performing a fast pruning step that significantly reduces the search space of association hypotheses. We evaluated the proposed algorithm on real sequences from WAMI imagery, and have shown that we obtain accurate tracking results in real-time and faster than the competition.

CHAPTER 5

Persistent Tracking

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. 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, including the one presented in the last chapter, are detection-based and rely on background subtraction or frame differencing to provide detections [63, 70, 67, 45]. Recent work shows detection based tracking approaches are powerful [64, 8], 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 chapter, 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 chapter is a multiple target tracking approach for wide area motion imagery that is better able to track targets through stops and brings us closer to persistent tracking. We evaluated the proposed approach on real sequences from wide area motion imagery and the results show increased object detection rate compared to competing approaches while keeping id switches and track breaks low.

5.1 Related Work

One of the earliest works on tracking targets in wide area imagery is by Perera *et al*. [63]. 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*. [90], 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. As we will see later, our proposed regression model is already able to correctly estimate the target state in the next frame without the need for spatial and road constraints.

Reilly *et al*. [70] 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*. [45], classic multiple hypothesis tracking on detections obtained with three-frame differencing is adopted. This presumably associates detections better than the frame-to-frame Hungarian algorithm, but may be more costly. Nevertheless, a real-time distributed architecture is presented. There is no provision for stationary targets.

In the last chapter we presented another approach that uses more than one frame of data to infer tracks. This approach uses long temporal windows of 8 seconds (16 frames) 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 [23, 53]. Doretto and Yao [23] 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. Without color, detection performance was much worse. In [53], 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 [78] 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 [93, 64, 8]. 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*. [88]. 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. For these reasons, we also adopt a regression model in our approach.

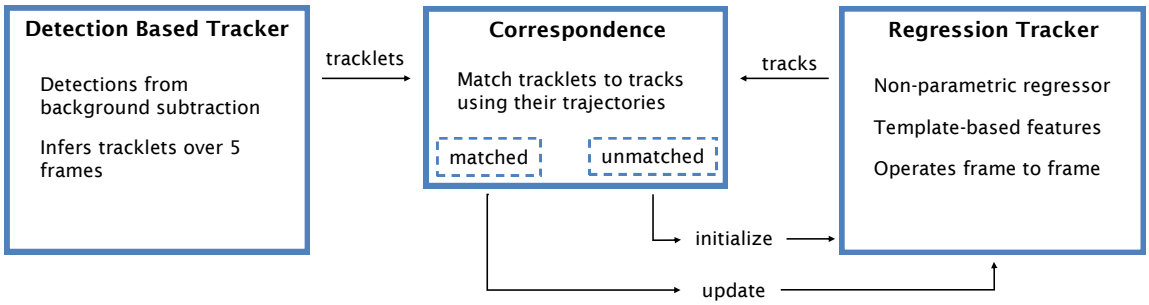


Figure 5.1: Overview of our approach. There are two trackers running in parallel complementing each other.

5.2 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, which is easier in the sense that the appearance variation is smaller, but more difficult in the sense that we need to deal with a small training set and have a limited computing budget when there are hundreds of targets to track. We discuss our appearance model in Section 5.2.1.2.

Whatever appearance model we choose, we need to be able to initialize it and update it when necessary (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, if any, 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 reasons for which will become clear soon.

The detection based tracker from the last chapter operates on a sliding window of 5 frames, and generates tracklets that correspond to moving targets every frame. These tracklets are matched to existing tracks maintained by the regression tracker. Tracklets that match an existing track are associated with it for potential model update. Tracklets that do not match an existing track are considered to be new targets, and are used to initialize the regression tracker, which then takes control over tracking them. We said potential model update, because the state and appearance model of a regression tracker is only updated when its failure is detected. This brings significant computational savings.

Our approach is illustrated in Figure 5.1. The key components are a detection based tracker, which is explained in the last chapter, a regression based tracker, explained in Section 5.2.1, and tracklet-track matching and failure detection, explained in Section 5.2.2. We now explain each of these components in detail.

5.2.1 Regression Tracker

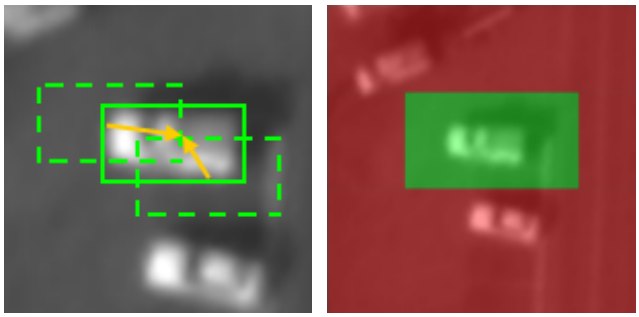
We are given a moving target's location in the first 5 frames and our task is to track the target as long as possible, even through stops. The usual way to proceed is to learn a classifier (appearance model) from the initial examples, predict (sample) the target's state in the next frame using a motion 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) [4, 5, 42]. 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. They move with constant velocity, with short bursts of acceleration when they slow down, turn, or speed up. We adopt the standard constant velocity motion model, but even for more complex models, there will be instances where its prediction is not going to be accurate. In those cases (vehicle accelerating when assuming constant velocity), we need to rely on the appearance model to find the true target state using the noisy set of samples.

The critical piece in this framework is the classifier, and specifically the set of examples used to train it. If it is a discriminative classifier and the positive examples are contaminated with a negative example(s), or the negative examples are contaminated with a positive example(s), accuracy of the classifier will degrade and the target's state is going to drift. Similarly, in the generative case, if the examples are not aligned or contain varying amounts of background, the classifier will learn the wrong function and the target's state is going to drift. This is a major

problem, which is why recent methods are careful about what examples are used to train the classifier [42], or optimize a function that is more robust to this contamination [5]. In our domain, where the target size is only 20×10, even a small error of a few pixels can cause the tracker to permanently lose track if the target is accelerating.

We argue that a binary classifier is not the appropriate way to model small transformations (translations, rotations) of the target. It is trained to make binary decisions, and when a test sample is not centered on the target, the target is arguably both "there" and "not there", and the classifier's decision is going to be suboptimal. Classifiers can also provide confidence or likelihood, but when we are sampling around our search region, there could be several possibilities with the same or similar confidence, making it difficult to choose the correct one just by searching for the maximum.

We believe that a much better way to proceed is to actually learn the effect of the displacement of target's state on appearance. In other words, we should learn the effect of target's translations or rotations on its appearance. This is possible when training a regressor rather than a classifier, and was first introduced by Williams *et al*. [88]. 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



(a) (b)

Figure 5.2: 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).

in contrast to a yes/no answer output by a classifier, which does not tell us as much. This is illustrated in Figure 5.2a.

The primary advantage of a regressor, as first noted by [88], 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. This comes under one condition though, and that is 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. The regressor always outputs a value, but the value in this case would be meaningless. In [88], 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 visible, a regressor is an inherently more powerful function than a classifier (much less likely to cause drift). This is illustrated in Figures 5.2 and 5.3.

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 observed to perform better than using an SVM as in [88].

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. In other words, when we do regression multiple times on different samples, we would expect a cluster to form around the true 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, with a regressor trained to output translations, we build a 2D histogram of the target state estimates, where the bins are of some small fixed size, such as 3×3. We then choose the bin with the highest number of samples as the best cluster. If the number of samples is higher than some threshold, these samples are then all considered valid. Alternate schemes to find the best cluster, or classify samples as valid/invalid can be used, but this approach has worked well for us (and better than an SVM).

Figure 5.3 shows an example on how this approach works when a vehicle is decelerating. The left side of the figure shows that the motion model assumes the target is moving faster

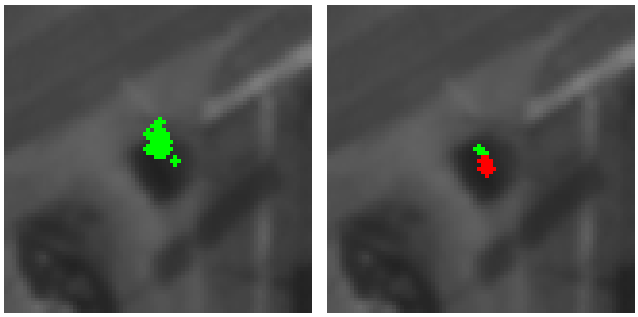


Figure 5.3: A regressor improves target state estimates. The left image shows samples before a regressor is applied. The right image shows the same samples after a regressor is applied. Crosses colored in red denote the final set of valid samples.

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.

5.2.1.1 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 [88]. Its advantage is that it is fully probabilistic, but

Algorithm 1 Regression Tracker

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

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

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

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

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

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

// reduce false alarms
check frame difference is consistent with est. motion
verify blob-like appearance

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

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 [33]. The regressor is of the form

$$f(\mathbf{x}_0) = \frac{\sum_{i=1}^{N} k(\mathbf{x}_0, \mathbf{x}_i)\mathbf{y}_i}{\sum_{i=1}^{N} k(\mathbf{x}_0, \mathbf{x}_i)} \tag{5.1}$$

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, such as $(\Delta x, \Delta y)$ when regressing target's translation or $(\Delta x, \Delta y, \Delta\theta)$ 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 [61], since our features are effectively probability distributions (see below). The kernel function is

$$k(\mathbf{x}_0, \mathbf{x}_i) = \exp(-\lambda KL(\mathbf{x}_0, \mathbf{x}_i)) \tag{5.2}$$

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.

5.2.1.2 Features for Regression

Some of the recent state of the art methods have relied on templates as their features of choice [42, 22]. 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 orientation of the initial 5 examples is estimated by fitting a quadratic function to the trajectory, and using the tangent at all positions as the target orientation. A quadratic function is used to handle tracklets that are initialized during turns. The bounding boxes of the initial examples can have different sizes, so we also estimate one fixed size of the target using least squares (given the target orientations and unoriented bounding boxes).

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. The shape mask is again expressed in the canonical frame. 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 5.4 illustrates the template learning process.

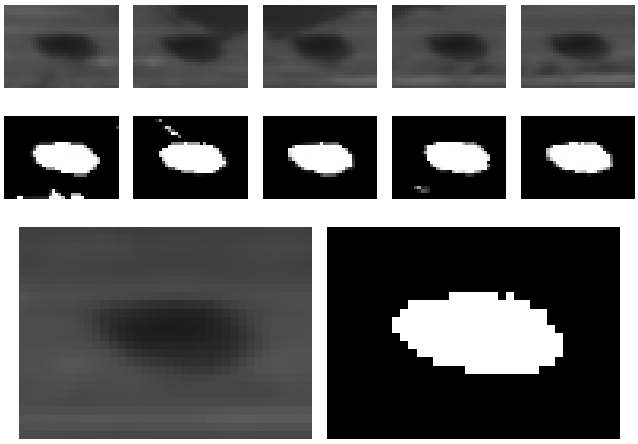


Figure 5.4: Target template as well as the shape mask is learned from 5 examples. The template is expressed in a canonical coordinate frame. Figure 5.3 shows the original target orientation and size.

Given this template, we would generate a training set for regression by displacing the template with known varying amounts of translation (and possibly rotation). See Figure 5.5 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 (and canonical images) 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

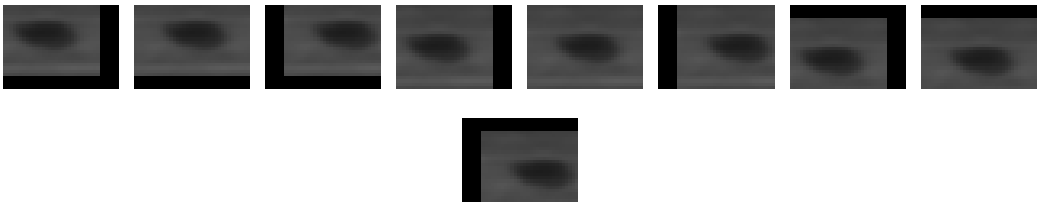


Figure 5.5: Subset of the displaced versions of the template used in regression. Each image has a corresponding shape mask.

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

$$\mathbf{x} = \left[g(\mathbf{I} - \mathbf{T}_0)^T g(\mathbf{I} - \mathbf{T}_0), \cdots, g(\mathbf{I} - \mathbf{T}_K)^T g(\mathbf{I} - \mathbf{T}_K) \right] \tag{5.3}$$

where

$$g(\mathbf{D}[j]) = \begin{cases} 10 & \mathbf{S}_i[j] \leq 128 \\ \min(|\mathbf{D}[j]|, 10) & \text{otherwise} \end{cases} . \tag{5.4}$$

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. In a way, we are counting the number of pixels that have the same intensity as the template, where "same" is defined as an intensity difference less than 10. Pixels outside of the shape mask are penalized equally, no matter what the difference in intensity. By normalizing this feature vector we get a probability distribution that could be used in our kernel function.

Since many of the elements of the feature vector have similar values (probabilities), to make the differences in the distributions more pronounced, we do an additional normalization. We subtract the minimum value, set those elements that would be 0 to a very small number, such as 0.0001, and renormalize to a distribution. This gives us a larger variation in the KL-divergence.

5.2.1.3 Motion Modelling

As mentioned earlier, we adopt a constant velocity motion model. 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 position of the target in the next frame is predicted by adding the current estimate of the velocity to the current position. We estimate target velocity using a standard Kalman filter.

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.

5.2.2 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 [67]. 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 blob appearance model. The blob appearance model is not

discussed here, but it is a relatively simple model based on a center-surround feature. 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. 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.

5.3 Results

We evaluated the proposed algorithm on sequences from a proprietary wide area aerial imagery dataset. The dataset was stabilized, georeferenced, and divided into several $2K \times 2K$ tiles before tracking. We selected two tiles for evaluation, each 2791 frames long, one with light traffic and few stopping cars, the other with heavy traffic and many stopping cars.

Several metrics were used to measure performance: object detection rate (ODR), false alarm rate (FAR), mean cumulative swaps of tracks (SWPS), and mean cumulative broken tracks (BRKS). Object detection rate is defined as the fraction of detections in the ground

	Proposed	[67]	[70] ($p = 1$)*
Object Detection Rate	0.74	0.69	0.76
False Alarm Rate	4.71	1.88	54.3
Track Swaps	1.64	1.61	8.15
Track Breaks	0.91	0.68	2.00
Frames Per Second	0.39	> 5	3.58

Table 5.1: Comparison of the proposed approach with competing methods on a tile with light traffic.

	Proposed	[67]	[70] ($p = 1$)*
Object Detection Rate	0.67	0.63	0.66
False Alarm Rate	27.5	27.3	137
Track Swaps	5.22	6.58	9.01
Track Breaks	2.04	1.54	2.65
Frames Per Second	0.07	> 2.5	2.35

Table 5.2: Comparison of the proposed approach with competing methods on a tile with heavy traffic.

truth found in the estimated tracks. False alarm rate is defined as the average number of false detections in estimated tracks in every frame. Mean cumulative swaps of tracks is defined as the average number of swaps (ID switches) in every ground truth track (over its lifetime). Mean cumulative broken tracks is defined as the average number of breaks in every ground truth track (over its lifetime). A break happens when a ground truth track is not matched to any ID in the next frame. These last two definitions are based on [71]. Computational efficiency was measured by the average number of frames processed per second (FPS). The results are shown in Tables 5.1 and 5.2.

We compared our approach to the one presented in the last chapter and [70]. *We were able to run [70] on only the first 1024 frames, which gives it some advantage. The object detection rate of the proposed approach is significantly better, indicating targets are being tracked longer, including through stops. The false alarm rate increased on the easier tile, but stayed the same

on the more difficult one. One of the factors behind false alarms is when the regression tracker loses track of the target and locks on to something in the background. When this happens, it's usually because the regression model was not initialized properly. The regression model is initialized using detections from background subtraction, and when a target is moving slowly, these detections can be noisy. As a result, 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.

The id switch rate (track swaps) is about the same in both approaches. This is because both approaches do not do any joint inferencing of target states. Since the states of targets are estimated independently, id switches are possible and not mitigated. Jointly tracking multiple targets with explicit constraints between them is the subject of our future work.

Track breaks are about 33% higher in the proposed approach. We believe this may be due to the weaknesses of the motion model. Either our covariance is too small for certain cases, or it is big enough, but we use a too small number of samples for cluster formation. If clusters do not form, our algorithm will declare it is too uncertain about the target state, and not estimate an observation for that frame. Subsequently, if there is no corresponding tracklet to recover from, or the logistic regression classifier makes the wrong decision, a break will be generated.

5.4 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 motion imagery show increased detection rates, with limited increases in false alarms and track breaks.

CHAPTER 6

Using 3D Scene Structure to Improve Tracking

In the previous chapter we have introduced a multi-object tracking algorithm that estimates trajectories of all moving objects in the video and have shown good results. Nevertheless, there still remain "corner" cases, which require a fundamentally different approach to be solved. One such case is when the moving object becomes occluded long enough for motion prediction to fail, or if the occlusion is short, but the object does not have a distinguishing appearance for successful reacquisition. These cases are difficult to solve without additional constraints. Yet, they occur often enough to warrant attention.

Again, we focus on tracking vehicles from a moving airborne platform in wide area surveillance, where the kind of occlusions just described happens when vehicles move behind buildings and other structures. The appearance of vehicles here is very weak, usually only allowing

Figure 6.1: Automatically computed occlusion map (right) for the image on the left.

us to distinguish light-colored vehicles from dark-colored vehicles, and small vehicles from large vehicles. Tracking vehicles across such occlusions is practically impossible without knowing more information. Here we explore the use of known 3D scene structure to estimate dynamic occlusion maps and improve tracking performance.

An occlusion map as used here is a binary map, which indicates what regions of the image are occluders of moving objects. See Figure 6.1 for an illustration. The occlusion map is not necessarily complete, there may be occluders which are not marked in the map (such as trees). Furthermore, the occlusion map may not be static for the duration of the image sequence due to camera motion. Importantly, the occlusion map is automatically computed from the video stream by estimating the camera pose in geo-coordinates and projecting a database of geo-referenced 3D models of buildings (occluders) to the image.

An occlusion map is useful, because it gives us the ability to detect occlusion events: a vehicle becoming occluded, a vehicle becoming visible. Analyzing these occlusion events, we can determine sources and sinks of traffic in the scene. While sources and sinks can be estimated

without the use of an occlusion map [82], such techniques are not robust enough to be used in our domain, where many false tracks arise due to the parallax motion of structures off the ground plane. By matching a sink with a source, we can apply an *ordering* constraint to solve the tracking problem: the sequence of vehicles becoming occluded should be approximately the same as the sequence of vehicles becoming visible. This sequence alignment problem is solved using dynamic programming. Note that by solving this problem we are able to correctly match vehicles before and after occlusion even if they stop or change motion while being occluded.

The primary contribution of this chapter is a novel use of 3D scene structure to improve tracking performance for objects moving through occlusions where motion prediction fails and/or when the appearance of objects is only weakly discriminative. We have evaluated the algorithm on real sequences from a publicly available wide area motion imagery dataset [50] and have shown that track fragmentation decreases and outperforms the Hungarian algorithm.

6.1 Related Work

The most related work to ours is the use of geo-spatial information to aid tracking [89, 90]. A recent example of this is work by Xiao *et al*. [89], where the camera pose is estimated in geo-coordinates to enable the assignment of semantic labels (building, tree) to image regions, and for road detection. The paper shows an increase in tracking accuracy when depth information, road network, and semantic scene segmentation is in use. However, there is only a a brief mention of using this information for occlusion handling in tracking, and no algorithm is

given. In the followup work [90], an improved scalable tracker using GIS road network is presented, but again there is no mention of occlusion handling.

Tracking objects through occlusions has been considered most recently by Kaucic *et al*. [44] and Perera *et al*. [63]. In the earlier work of Kaucic, an occlusion map is estimated by segmenting the image into regions and training a classifier to label each region as one of a "building, road, grass, or vehicle." This occlusion map is then used to constrain location of image features for video stabilization, filter out false moving object detections, and detect when objects are likely to become occluded. One disadvantage of computing an occlusion map using image features is that it is generally not as accurate as one computed using known 3D scene geometry. Tracks before and after occlusion are linked using the Hungarian algorithm, where the pairwise cost is based on appearance similarity (color histogram matching) and motion prediction (constant velocity model).

Perera *et al*. extend this work by incorporating merge and split hypotheses to handle the many-many correspondence between moving region detections and objects. The Hungarian algorithm is again used as the computational engine for determining correct track associations. Template matching is used to define an appearance cost and a linear motion model to define a kinematic cost.

Besides the Hungarian algorithm, another class of methods for tracking occluded objects is based on a tracking context [1, 91, 28]. Our work also falls in this category as tracking context is implicitly formed when sequences of tracks are matched against other track sequences. In [1], the context is other moving objects, called predictors, whose motion correlates with

that of the occluded object. The predictors are used to predict the location of the occluded object and associate it with the correct track upon reacquisition, and are found using Lyapunov Characteristic Exponent. A more sophisticated approach is presented in [91], where an object is continuously tracked collaboratively with auxiliary objects (the context), and the relationship between the auxiliary objects and the object of interest is modeled using a graphical model. Finally, most recently Grabner *et al*. presented an approach where the context of an object is a set of "supporters," which vote for the location of the object being tracked. As long as the supporters are visible, the location of the occluded object can be accurately determined.

The task of the driving algorithm in this work is to align two sequences of tracks. Sequence alignment is a well-known pattern recognition technique that has mostly seen its use in speech [72], and gesture (action) recognition [20, 10] to temporally align two sequences and in stereo correspondence to estimate disparities [76]. The classic method is Dynamic Time Warping (DTW), which is a dynamic programming algorithm to find the optimum non-linear alignment of two sequences. To our knowledge, sequence alignment has not been used before to track objects through occlusions.

6.2 Approach

The goal of our work is to improve tracking of moving objects, especially when they become occluded for a significant amount of time. We focus on the case where the occlusion is long

enough to cause an object's track to break and become fragmented. The output of our algorithm is a track correspondence, which indicates what tracks should be merged or linked with other tracks.

As indicated in the introduction, we are concerned with tracking vehicles from a moving platform in wide area surveillance. Here, the appearance of vehicles is not very discriminative, and cannot be solely relied upon for correct track linking. Therefore, as has been done before, we combine appearance with a kinematic model, at least when the kinematic model is informative. Actually, the strength of our method is primarily evident when the kinematic model is unable to correctly predict a vehicle's location after occlusion.

We are able to accomplish this by applying an ordering constraint, which says that the sequence of vehicles becoming occluded is approximately the same as the sequence of vehicles becoming visible after occlusion. In order to apply this constraint, we need to be able to reliably detect sources and sinks of traffic in the imagery and match them up correctly. A traffic sink is where vehicles become occluded and a traffic source is when they become visible again. Dynamic occlusion maps are essential for a robust estimation of these sources and sinks. The occlusion maps are computed automatically by estimating the camera pose in geo-coordinates and using a database of geo-referenced 3D models of buildings. These 3D models can be computed from imagery using dense 3D reconstruction techniques, or can be acquired from online sources [27] as we have done in this work.

A flowchart of our approach is in Figure 6.2.

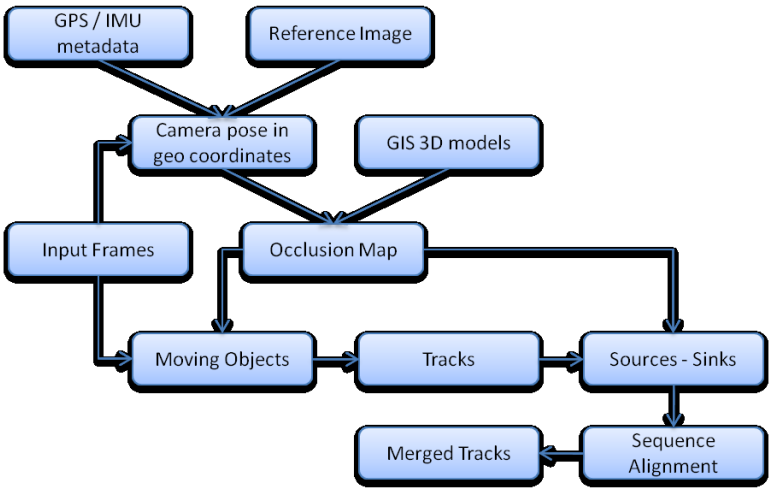


Figure 6.2: Our framework for tracking vehicles across long occlusions.

6.2.1 Estimating the Dynamic Occlusion Map

The dynamic occlusion map is estimated by projecting a database of geo-referenced 3D models of buildings to the input video stream. Every pixel that is written during rasterization is marked as occluding. The occlusion map needs to be in agreement with the underlying imagery, but we do not require perfect registration. Accurate estimate of the camera pose in geo-coordinates enables us to achieve good registration.

GPS and IMU metadata associated with the input imagery provide a good initial estimate of the camera pose, but can not be used directly. To refine this initial estimate, we register the input imagery with a reference image (map). The input image transformation is modeled by a homography. We perform image registration in a fairly standard manner: by hierarchically estimating the image transformation, incrementally increasing the transformation complexity, using state of the art image features, and using a robust model estimator.

First, a Gaussian image pyramid is built for both the reference image and the video image. The images are first registered at the top of the pyramid, then the images at the next level down are warped using the just estimated transformation, registered again, and the process continues all the way down to the bottom of the pyramid which is at the highest resolution. This hierarchical process significantly increases the speed and accuracy of the registration, because at each level, feature correspondences only need to be found in a small local neighborhood.

The image transformation complexity is simple at the top of the pyramid and increases on the way down. More specifically, at the top levels of the pyramid, only image translation is estimated, and a full homography transformation is not estimated until the bottom levels. This is important, because at the top levels of the pyramid, the image resolution is not high enough to support a complex image transformation, so the estimation procedure is unstable.

State of the art image feature descriptors are used to establish correspondences at every level of the pyramid [86]. These are accurate and fast to compute. Finally, when estimating image transformation, RANSAC is used to provide a robust estimate. This is necessary to avoid using image correspondences on buildings and only make use of correspondences in planar areas.

Once we have a set of correspondences between the reference map and the video image that satisfy the final homography estimate, we solve for the camera pose by assuming all the correspondences lie on a plane. The algorithm presented by Schweighofer and Pinz [77] is used for this step. It provides a robust estimate of the pose by explicitly avoiding a solution that is a local minimum of the objective function.

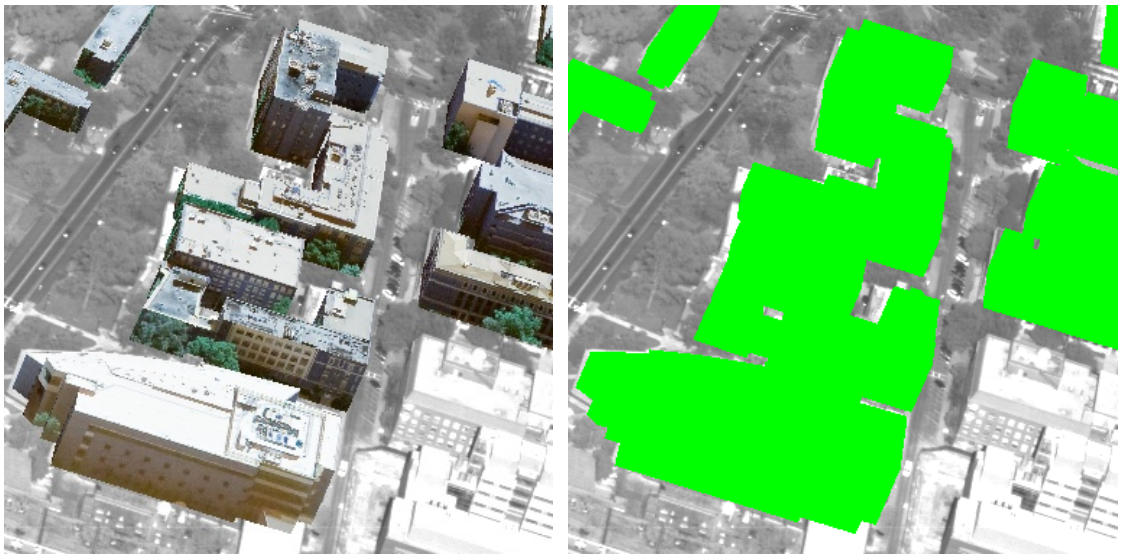


Figure 6.3: Example of the accuracy of the camera pose estimation. Accurate estimate of the camera pose results in a good registration of the 3D models (left) and occlusion map (right) with the image sequence.

The result of occlusion map estimation is illustrated in Figures 6.1 and 6.3. The average registration error is 10 pixels.

6.2.2 Tracking

Tracks of vehicles in the image sequence can be estimated by any multi-object tracking algorithm. By taking advantage of the just computed occlusion map, the number of false tracks can be significantly reduced. In our implementation, we used a tracker introduced in the previous chapter.

6.2.3 Estimating sources and sinks

Our goal is to track vehicles even when they become occluded for a long time. Such long occlusions cause a tracking algorithm to lose a vehicle's track and initiate a new one when the vehicle becomes visible again. To achieve our goal, we need to find these false initiations and merge them with the correct earlier tracks that were prematurely terminated.

False initiations due to occlusion form a traffic "source" -- a location where many tracks originate. Premature track terminations due to occlusion form a traffic "sink" -- a location where many tracks stop. Of course, it is possible for a single vehicle to become occluded and reappear again. However, these locations may be difficult to identify with high confidence without other information, such as the road network. Fortunately, we can easily work around this problem by just waiting long enough for more tracks to enter the source and leave the sink.

The dynamic occlusion map is essential in detecting sources and sinks reliably. Source detection begins by taking all tracks that we have estimated (by the end of the sequence or a sliding temporal window), and computing the distance from the starting point of the track to the nearest occluder pixel. If this distance is "small," we record a leave-occlusion event at the track's starting location. The definition of "small" depends on the accuracy of the estimated occlusion map. In our work, we have used a value of 20 pixels (15 meters on the ground).

All the leave-occlusion events are then clustered in the (x, y, x', y') space, where (x, y) are the image coordinates of an occlusion event, and $x' = x + c \cdot v_x$, $y' = y + c \cdot v_y$, where v_x, v_y is the moving direction of the associated track. The constant c is used to give an appropriate

scaling to the direction vector, and it is set to 20 in our experiments. Agglomerative clustering is used for this task: initially a cluster is created for each event, and the closest clusters are repeatedly merged until the distance between them reaches a maximum. The distance between two clusters is the Euclidean distance between the cluster centers. The only parameter of this clustering technique is the stopping criterion. In our work, we have used a value of 20 pixels. Note that due to the moving camera, the occlusion map is dynamic, and the occlusion events, while localized, will not be generated at a single point. The resulting clusters are the desired traffic sources.

Sink detection proceeds similarly, the only difference being that now we compute distance from the ending point of a track that has been declared terminated by the tracking algorithm. If this distance is small, we record an enter-occlusion event. After clustering these events we get the desired traffic sinks.

For efficiency, a distance transform is computed for an occlusion map. All the required distance calculations then only require a look up operation. To put this procedure into perspective, in a 801x233 sequence of 200 frames, we recorded 50 leave-occlusion events and 35 begin-occlusion events. After clustering, 4 sinks and 7 sources are produced. See Figure 6.4 for an illustration.

6.2.4 Source-Sink correspondence

After estimating sources and sinks, we need to know which sink corresponds with which source. Knowing this correspondence, we can then take the sequence of tracks entering the

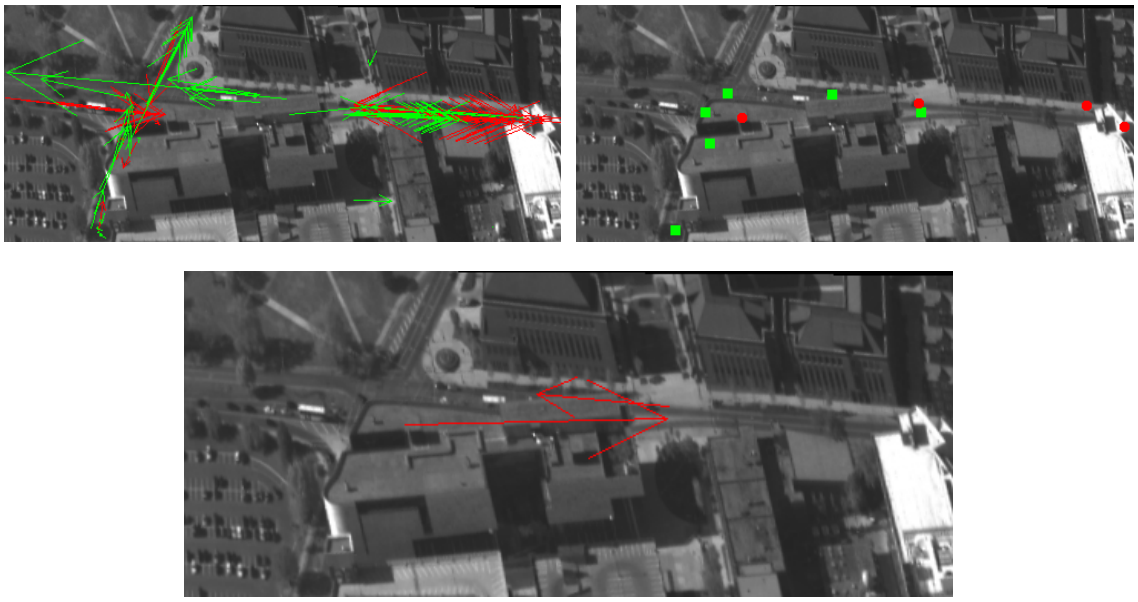


Figure 6.4: Example of source and sink detection. The top-left figure shows all occlusion events and the direction of travel for each one. The top-right figure shows the resulting clusters -- sources and sinks. Red circle indicates enter-occlusion and green square indicates leave-occlusion. The bottom figure shows correspondences between sources and sinks.

sink (becoming occluded) and align it with a sequence of tracks leaving the corresponding source (reappearing). The difficulty of this correspondence problem depends on the complexity of the occlusion. An occlusion where one sink is connected to more than one source (a one-many relationship), is considerably more difficult to handle than an occlusion with a one-one relationship between sink and a source.

In this work, we assume a one-one correspondence between sources and sinks, which is often satisfied in aerial surveillance imagery over urban areas. This does not mean that *all* vehicles entering a sink must leave a corresponding source. It is possible that a vehicle parks behind the occluding building, or more simply that a tracker does not track every vehicle

entering / leaving the occlusion. However, we do expect that *most* vehicles entering a sink will in fact reappear at the source.

Using this assumption, we find a corresponding source for every sink by setting up a weighted bipartite graph matching problem, and solving it using the Hungarian algorithm. The pairwise matching similarity is defined as:

$$c_{ij} = C_1 * (d_1 + d_2) + (C_1/d) + C_2 * s(i) * s(j) \tag{6.1}$$

where d_1 is the dot product of the direction of traffic entering the sink with the direction of traffic leaving the source, d_2 is the dot product of the direction of sink to source with the direction of traffic entering the sink, d is the distance from sink to source, $s(i)$ is the number of occlusion events (tracks) in cluster i, and C_1, C_2 are constants that weight each factor appropriately and make the matching similarity an integer. In our experiments, $C_1 = 1000, C_2 = 100$. The Hungarian algorithm will return an assignment of sinks to sources that maximizes this similarity. Furthermore, we require that the angle between the direction of traffic entering the sink and leaving the source is less than $60°$. A source-sink pair that does not satisfy this is removed from consideration. See Figure 6.4 for an illustration.

6.2.5 Sequence alignment

With an established correspondence between a sink and a source, our task is to match the sequence of tracks at the sink to the sequence of tracks at the source. By matching two sequences,

and not two sets (as is customary with a Hungarian algorithm), of tracks we are applying an additional ordering constraint to solve the track linking problem. Using this additional constraint is essential when motion prediction fails (due to long occlusion) and/or when the appearance of the tracked objects is not very discriminative.

To solve the sequence alignment problem, we use a standard dynamic programming algorithm that is similar to Dynamic Time Warping and has been extensively used in computational biology. Let the sequence of tracks at the sink be $X = x_1 x_2 \cdots x_m$, and let the sequence of tracks at the source be $Y = y_1 y_2 \cdots y_n$. An alignment of the two sequences is a set of ordered pairs of tracks in X and Y such that there are no "crossing" pairs. In other words, if $(i, j), (i', j')$ is in our solution, and $i < i'$, then $j < j'$ [48]. This often holds in practice. To handle the case of a vehicle becoming out of order during the time it is occluded, we allow some tracks in X and Y to not match at all. For each such "gap," a penalty of δ is incurred. For every pair of tracks that do match, there is a matching cost $\xi_{x_i y_j}$. The cost of an alignment is the sum of its gap and matching costs. A dynamic programming algorithm minimizes this cost with the following recurrence [48]:

$$f(i, j) = \min[\xi_{x_i y_j} + f(i-1, j-1), \delta + f(i-1, j), \delta + f(i, j-1)] \tag{6.2}$$

In our domain, the matching costs ξ are computed from 4 terms, appearance term α, size term β, kinematic term γ, and feasibility term τ:

$$\xi_{x_i y_j} = \tau(x_i, y_j) * \gamma(x_i, y_j) * (\alpha(x_i, y_j) + \beta(x_i, y_j)) \tag{6.3}$$

The feasibility term is defined as

$$\tau_(x_i, y_j) = \begin{cases} 1 & \text{link } y_j \text{ with } x_i \text{ is feasible} \\ \infty & \text{otherwise} \end{cases} \tag{6.4}$$

where linking y_j with x_i is feasible if y_j becomes visible at a time after x_i becomes occluded, and given the distance and time between x_i and y_j, if the acceleration from x_i to y_j is physically possible ($\leq 10m/s^2$).

The kinematic term is defined as

$$\gamma(x_i, y_j) = \min(|t'_j - t_j|, G)/G \tag{6.5}$$

where t'_j is the predicted frame number of track x_i at the first location of y_j, and t_j is the actual frame number of track y_j at its first location. By clamping this time difference to G, the kinematic cost becomes unimportant for sequences of tracks which stop behind an occlusion, or otherwise have unpredictable motion. At the same time, if the motion prediction succeeds, the appearance and size terms become unimportant. The parameter G is set to 8, which corresponds to a time difference of 4 seconds in our imagery.

The appearance term is defined as

$$\alpha(x_i, y_j) = \min(D(h(x_i), h(y_j)), A)/A \tag{6.6}$$

$$D(p, q) = 0.5 * (D_{KL}(p||q) + D_{KL}(q||p)) \tag{6.7}$$

where D_{KL} is the Kullback-Leibler divergence, and $h(\cdot)$ is a probability distribution of image intensity on the foreground of the object. This distribution is calculated from a histogram of 16 bins. The parameter A is estimated as the maximum value of $D(p, q)$ in a training set of 11,208 appearance term computations. In our experiments, this value is 3.5.

The size term is defined as

$$\beta(x_i, y_j) = \min(|s(x_i) - s(y_j)|, B)/B \tag{6.8}$$

where $s(\cdot)$ is the median size of the tracked object. The parameter B is estimated as the maximum value of $|s(x_i) - s(y_j)|$ in a training set of 11,208 size term computations. In our experiments, this value is 17.

The parameter δ controls the likelihood of tracks becoming out of order. When it is 0, tracks before and after occlusion can be matched with an arbitrary number of gaps. When it is high, the first track entering an occlusion will always match the first feasible track leaving an occlusion. In our experiments, we have empirically determined the value of δ to be 0.40.

The alignment of tracks which minimizes the cost function $f(i, j)$ is the solution to our tracking problem. That is, by merging those tracks the vehicle continues to be tracked after the occlusion.

6.2.6 Computational Complexity and Implementation

The computational complexity of the sequence alignment algorithm is $O(nm)$, where n, m is the number of tracks in a sequence. In practice, this number is often small, so the algorithm is quite efficient. We have implemented the algorithm just presented in C++.

6.3 Results

We have evaluated the proposed track linking algorithm on sequences captured from an airborne sensor. The sequences come from the publicly available CLIF 2006 dataset [50]. This dataset is captured at roughly 2 Hz, and it is in grayscale. It is an example of persistent surveillance imagery, where the airplane makes several circular flyovers around the campus of Ohio State University.

Our database of geo-referenced 3D models has 90 models of buildings that were downloaded from Google Earth. For the geo-registration step of our camera pose estimation, we have used a reference image from USGS [83]. Camera calibration was achieved by manually selecting correspondences and performing bundle adjustment.

A baseline tracker (Chapter 4) was used to estimate tracks in 5 sequences. The sequences are 150-200 frames long each, and roughly 500x200 pixels in size. We have selected those subregions of CLIF for which 3D models are available. In sequences 1 and 3, vehicles makes a stop while being occluded. Fragmented tracks in all sequences were manually identified in the tracking output. After running the proposed algorithm on each sequence, we manually verified the correctness of the track linking suggested by the algorithm. Ideally, all of the

	Fragmented Tracks	Proposed Algorithm		Hungarian Algorithm	
		Correctly Linked	Incorrectly Linked	Correctly Linked	Incorrectly Linked
Seq 1	9	5	2	2	7
Seq 2	4	3	3	3	4
Seq 3	3	1	6	1	7
Seq 4	2	1	2	1	3
Seq 5	6	4	1	3	3
Sum	24	14	14	10	24

Table 6.1: Quantitative evaluation of the proposed track linking algorithm.

fragmented tracks identified earlier would be correctly merged together. The results can be seen in Table 6.1.

We compared the performance of the proposed sequence alignment to the widely used Hungarian algorithm. The matching cost used in the Hungarian algorithm was exactly the same as the one used in sequence alignment, but scaled and rounded to an integer value. The results of this comparison can also be seen in Table 6.1.

Examples of linked tracks in the sequences we tested can be seen in Figure 6.5, 6.6, 6.7, and 6.8.

The results show that the proposed algorithm is effective in linking tracks through occlusions. More than half of the fragmented tracks were successfully linked together. In contrast, the performance of the Hungarian algorithm is quite poor, with only a small number of tracks correctly linked and with a significant number of ID switches (incorrect links). Because we are using the same matching cost between tracks in both the sequence alignment problem and in weighted bipartite graph matching, we conclude that the ordering constraint is in fact responsible for the increased performance.

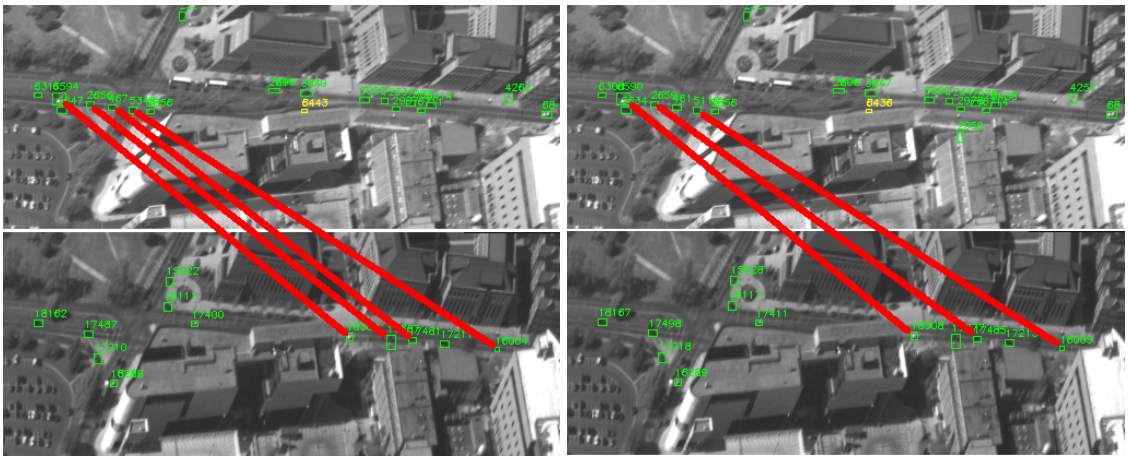


Figure 6.5: Linked tracks by the proposed algorithm (left) and Hungarian algorithm (right). Note that in this sequence, the vehicles stopped while being occluded. The only correct link by the Hungarian algorithm is the leftmost one. The top row shows vehicles before occlusion and the bottom row shows the same vehicles after occlusion.

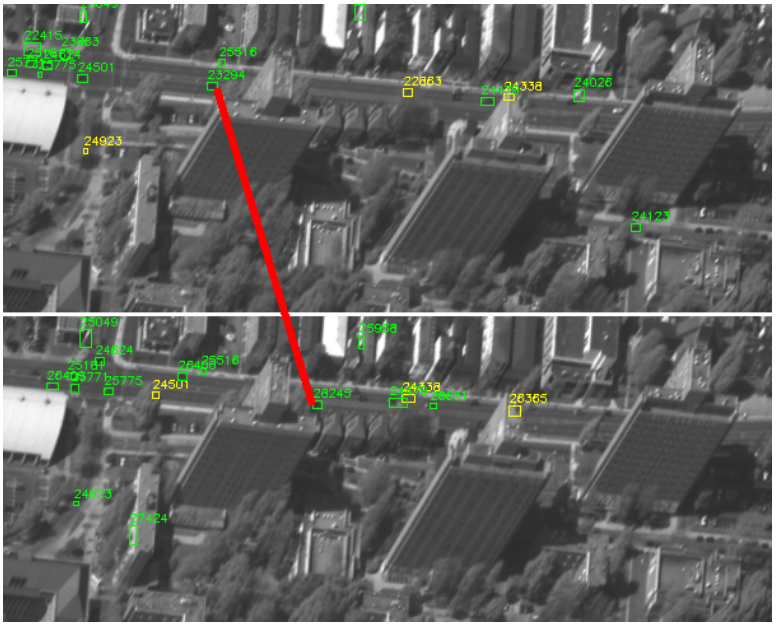


Figure 6.6: Correctly linked track by both the proposed algorithm and Hungarian algorithm. This is a short occlusion case, where motion prediction successful. This shows the algorithm works just as well in these cases. The top row shows the vehicle before occlusion and the bottom row shows the same vehicle after occlusion.

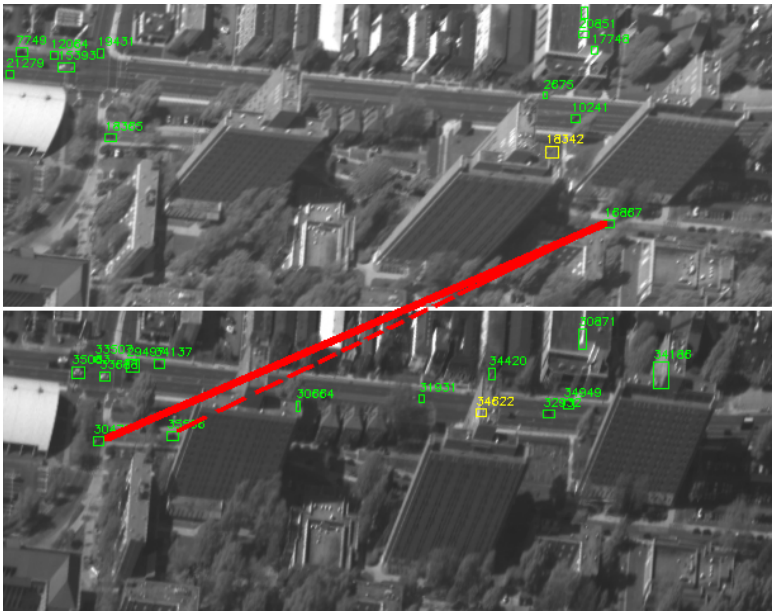


Figure 6.7: An example where both algorithms fail to link with the correct track, which is indicated with a dashed line. The top row shows vehicles before occlusion and the bottom row shows the same vehicles after occlusion.

Figure 6.8: Another example of a linked track by the proposed algorithm. The top row shows the vehicle before occlusion and the bottom row shows the same vehicle after occlusion.

The most common type of error in the solution to the sequence alignment problem is a track paired with a track that is next to the correct one (either ahead by one, or earlier by one). This can happen when the tracker does not track all vehicles entering or leaving an occlusion, and there is an appearance ambiguity as to which vehicle is the correct match. For example, when two dark colored cars are adjacent in a sequence, it is not clear how to choose the correct one. This does not happen when the appearance of vehicles is sufficiently diverse. In fact, the variation in appearance of vehicles acts like a synchronization mechanism -- it prevents the error in one assignment from accumulating and throwing off the whole sequence.

6.4 Conclusions

Our primary contribution is a novel use of 3D scene structure to improve tracking performance for objects moving through long occlusions, and it is especially suitable when motion prediction fails and/or when the appearance of objects is only weakly discriminative. The key feature of the proposed solution is the application of an ordering constraint, which stipulates that the sequence of tracks becoming occluded is approximately the same as the sequence of tracks becoming visible. A dynamic programming algorithm is used to solve the sequence alignment problem. The algorithm was evaluated on a vehicle tracking task in an aerial surveillance video. The results show excellent performance in terms of decreasing track fragmentation and outperform the Hungarian algorithm.

CHAPTER 7

Activity Recognition

The ability to automatically or interactively infer meaningful activities and events from large volumes of existing video data should be of considerable help to analysts. The goal of this chapter is to provide an activity recognition framework for wide aerial video surveillance, which is characterized by a very large field of view, large camera motion, strong parallax, a large number of moving objects, and a small number of pixels on targets [70].

We start with geo-registered tracks inferred by a tracking module after stabilization. Activities are defined as tracks associated with properties of one or more objects. Since an activity may involve a sequence of motion patterns (events) and multiple actors, how to represent events and activities is a challenging task.

We propose to define and recognize a large number of activities with the ERM (Entity Relationship Models) [14] framework, taking into account uncertainties associated with observations. The ERM is an appropriate framework to capture multiple relationships between

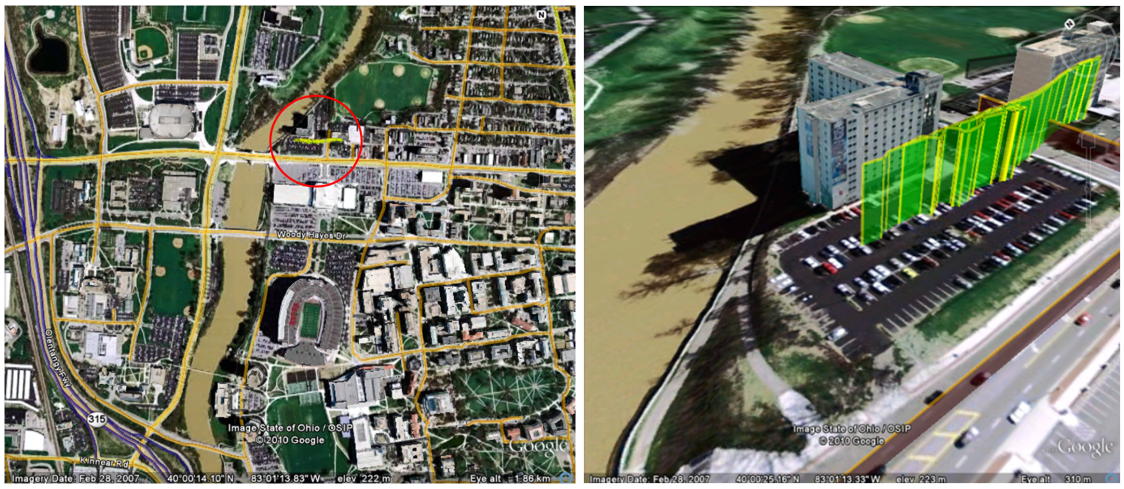


Figure 7.1: (A) We can identify a "source of tracks" by finding a set of tracklets that have the same starting location in different time periods. (B) rendered closed-up view, using Google Earth, shows that the location is a parking lot.

elements, which allows us to efficiently represent hierarchical structures, multiple actor activities, and context information. We use a RDBMS (Relational DataBase Management System), such as Microsoft SQL [60], to store and retrieve all meta-data in our activity recognition system, including tracking results, geospatial objects and context information, and use Structured Query Language (SQL) [60] to define and recognize activities. In this approach, finding an activity is equivalent to sending a set of SQL statements to the RDBMS. Also, RDBMS scales well with respect to the number of distributed systems, for a large amount of data.

Many activities can only be inferred within the context of geospatial information and the ERM framework is ideally suited to incorporate Geographic Information System (GIS) data. We use open geospatial data from Open Street Map [16] to extract geospatial objects, such as road networks, road types, and parking lots. To represent geospatial activities, we link visual tracks and geospatial objects.

Our contributions can be summarized as follows:

- We propose to use the entity relationship model (ERM), a well-established methodology in real world applications, to design and implement an activity recognition system for wide aerial video surveillance where vehicular segmented tracks are the essential components. By leveraging the powerful computational features of a RDBMS, which is an implementation of ERM, finding an activity is equivalent to sending a query to the RDBMS.

- We demonstrate that different types of activities, with hierarchical structure, multiple actors, and (geo) context information, can be effectively and efficiently defined and inferred using the ERM framework, taking into account uncertainties associated with observations.

- We also show that visual tracks can be interpreted as activities using geo information. By incorporating reference imagery and extensive using of GIS context, tracked objects can be associated with physical meanings, and several high levels of reasoning, such as traffic patterns or abnormal activity detection, can be performed.

The derived information is integrated into an existing visualization software for GIS (Google Earth [27]), which provides the capability to analyze and infer higher levels of activities, as well as serves as an interactive visualization system. Figure 7.1 shows an example of inferred results from real visual tracks. Experimental results on real tracks and GPS tracks validate our approach.

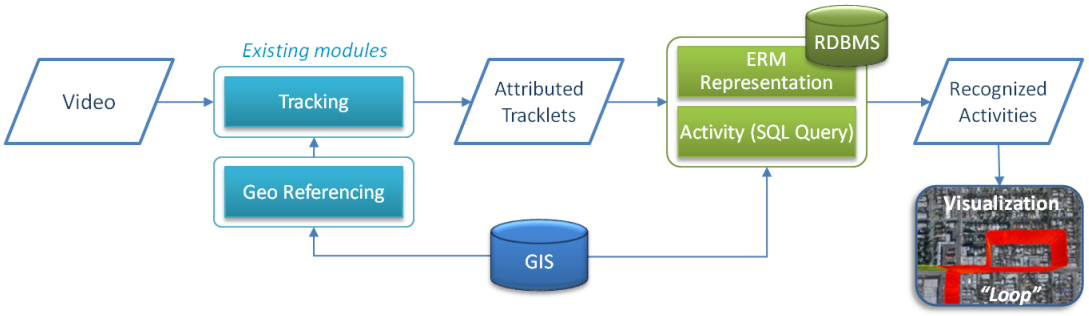


Figure 7.2: Overview of the proposed approach.

7.1 Related Work

We now present an overview of methods to represent activities. A recent survey [87] describes actions, simple motion patterns usually performed by a single actor, represented by non-parametric (e.g. template matching and dimensional reduction), volumetric (e.g space-time filtering and tensors), and parametric (e.g. HMMs and linear dynamic systems) methods. Activities, which are complex sequences of actions, are represented using graphical models (e.g. Dynamic Bayesian Nets and Petri nets [21]), Syntactic (e.g. Context Free Grammars and Attribute Grammars), and Knowledge Based (Constraint Satisfaction, Logic Rules, and Ontologies) approaches.

Classical pattern classification formulations, employing a fixed dimensional input vector, cannot handle complex activities which need to consider temporal relationships and context information. HMM-based approaches are widely applied to speech recognition to recognize a sequence of features. However, the assumption of Markovian dynamics and the time-invariant nature of the model restrict the method to simple stationary temporal patterns [87].

Graphical models, including Bayesian networks (BN) and Dynamic belief networks (DBN), have been widely applied for higher level representation and reasoning since Bayesian networks present conditional dependencies between random variables. Clearly, probabilistic inference provides better decisions from uncertain input data. However, how to acquire reasonable prior and conditional probabilities as prior domain knowledge is a critical issue for a dynamic environment. Current methods rely on the closed world assumption, which restricts this approach to well-defined, closed domains. For a large scale, real world problem, defining and learning the structure of the DBNs is very difficult because of the large number of variables with complex dependencies, and the need for very large amounts of training data.

Image interpretation is considered to generate a comprehensive stochastic grammar that can interpret an image by AND-OR graph that is equivalent representation to context-free grammars [30]. Because deterministic grammars (e.g. FSA, CFG) cannot code low-level uncertainties, Stochastic Context Free Grammars (SCFG) are presented for complex activities [38]. The limitation of SCFG is that complex temporal constraints, such as parallelism, overlap, and synchrony, cannot be expressed. For example, the rule $A \rightarrow abB$ does not specify whether the events can overlap or not, as mentioned in [18]. Damen proposes attribute grammars [19] to constrain the spatial relationships in visual scenes although the method is able to handle a single event.

Logic-based approaches rely on formal logic rules to describe activities related with common sense knowledge [59]. `Video Event Recognition Language' (VERL) is proposed based

on an ontology and first-order logic [25]. Fung [26] presented a combination of predicate logic and probability for information fusion and decision support.

We now briefly review the literature on activity in wide area surveillance. Recently, Reilly *et al*. [70] shows object detection and tracking in a wide area surveillance domain, where bipartite graph matching and linking tracks were applied to detection results, and grid cells were employed to provide a set of local scene constraints such as road orientation and object context for tracking. While many researchers have shown reliable tracking results, there are only a few works on activity detection and recognition for wide area videos. Pollard *et al*. [65] presented activity detection results using a complex probabilistic framework but only single activity, convoys, was presented and geospatial constraints were not considered.

Given our domain, where vehicular segmented tracklets are the essential components, we believe that activities can be effectively and efficiently inferred using a relational database model, coupled with a probabilistic framework to handle uncertainty. This is described in the next sections.

7.2 Approach

7.2.1 Problem Statement

The input to our system is a set of tracks $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, \cdots, \mathbf{o}_N\}$, where N is the total number of tracks. Each track contains M tracked points $\mathbf{o} = (p_1, p_2, \cdots, p_M)$. In activity recognition, multiple tracks can be associated with a single activity and a single track can contribute to

multiple activities. Hence, it is a many-to-many mapping from a set of input data to a set of class symbols. The activity recognition suggested in this chapter is defined to automatically find a subset of the input data ($\mathbf{O}$), which matches an activity $\mathbf{a}_j$,

$$(\mathbf{O}, \mathbf{a}_j) \rightarrow \{\mathbf{o}'_1, \mathbf{o}'_2, \cdots, \mathbf{o}'_N\} \tag{7.1}$$

where $\mathbf{o}'_i$ is a subset of the track $\mathbf{o}_i$. Figure 7.2 shows an overview of our approach.

7.2.2 Estimating tracks

Recognizing meaningful activities from vehicle tracks requires that the estimated tracks be useful -- long enough to capture interesting motion patterns. Any tracking algorithm that is robust enough to provide such tracks can be used in our work. In our implementation, we use a tracker that was introduced in Chapter 4.

7.2.3 Tracklets from the tracking module

A tracklet is the atomic spatio-temporal information, a segmented portion of a track, representing a vehicle's motion. Each tracklet has a collection of attributes $x_i = \{\lambda_1, \lambda_2, \cdots, \lambda_m\}$, where an element λ_i presents a physical property such as time, location, and speed. To determine these track segments, we note that, clearly, the most important points of the track are those where the direction of travel changes. Between these points, the motion pattern is constant and predictable. Therefore, our goal is to segment the track's trajectory into segments which are accurately approximated by lines (linear model). Points on the trajectory between

these segments are where the direction of travel changes, often as a result of the vehicle making a turn. Before segmentation, we filter out noise in the trajectories output from the tracker by minimizing a robust cost function (Huber) over a sliding window of 5 locations.

We determine the trajectory segmentation optimally, using a classic dynamic programming algorithm "segmented least squares" [7]. Given an ordered set of points $p_1, \cdots, p_j$ on the trajectory, the algorithm minimizes the cost function,

$$f(j) = \min_{1 \leq i \leq j} (\epsilon_{i,j} + \Theta + f(i-1)), \tag{7.2}$$

where $f(0) = 0$, and $\epsilon_{i,j}$ is the least squares error of fitting a line through points $p_i, \cdots, p_j$. Before the optimization, the trajectories are resampled so that the distance between adjacent sample points is constant (≈ 4 meters). The parameter Θ of this method is used to tune the tradeoff between the number of segments produced and the resulting error. In our implementation we have fixed the value of this parameter to 1, which we have observed to generate good segmentation results.

Tracklets are determined from the resulting segmentation by creating one tracklet for each segment ("straight" tracklet), as well as one for the path between every two adjacent segments ("turn" tracklet). Furthermore, straight tracklets longer than 100 meters are broken into shorter 50 meter segments. Now, for each tracklet, we compute a collection of attributes, such as location, heading (applies to straight tracklets), heading change (applies to turn tracklets), speed, acceleration, and accumulated distance traveled so far.

Entity
track point, tracklet, track, traffic rule,
road, building, area, $\cdots$
Relationship
tracklet -is on- road
road -has- traffic rule
must_stop -is a- traffic rule, $\cdots$
Event can be represented by a relationship
tracklet (track_id, $\cdots$, speed=95)
tracklet (track_id, $\cdots$, road_id)
road (road_id, $\cdots$, speed_limit)
speeding: tracklet.speed $>$ road.speed_limit

Table 7.1: ERM representation

7.2.4 Activity Representation Using ERM

We use ERM (Entity Relationship Models) to capture multiple relationships between elements. Such a framework has been extensively used and validated for a long period of time in real world applications [15, 14]. The basic entity-relationship modeling approach is based on describing data in terms of the three parts: entities, relationships between entities, and attributes of entities or relationships. The relationships include "Belonging to", "Set and subset relationships", "Parent-child relationships", and "Component parts of an object".

Hence, we represent track points $\{p\}$, tracklets $\{x\}$, and tracks $\{\mathbf{o}\}$ as entities and link the three entities: $\{p\} \subset \{x\} \subset \{\mathbf{o}\}$. The collection of physical properties of each tracklet is represented as the attributes of the tracklet entity (a RDBMS table).

We also represent geospatial data (traffic rules, roads, buildings, and areas) the same way. An entity "road" is a collection of road segments and each segment has a set of attributes such as type, name, and speed-limit. Table 7.1 illustrates our ERM representation.

An activity a_j is defined as a collection of tracklets obeying certain properties: $a_j = \{x | x \in \Omega_j, C_j(x) > \theta_j\}$, where Ω_j, $C_j(x) \in [0, 1]$, and θ_j represent the relationship associated with the activity, the confidence function and the recognition threshold, respectively. The relationship Ω_j links between the attributes of entities, which include both the physical properties of tracklets and the geospatial data.

Thus, *"Speeding"* can be seen as an activity defined by the relationship between two attributes of tracklets (e.g. speed) and geospatial objects (e.g. speed-limit):

$$\begin{aligned} speeding := \{x| \quad & r \in \mathcal{G}_{road}, x.roadID = r.ID, x.s > r.s, \\ & C(x.s, r.s) > \theta\} \end{aligned} \tag{7.3}$$

where $r, r.ID, x.roadID, x.s, r.s$ represent a road from GIS data ($\mathcal{G}_road$), its ID, the road ID of x, the speed of x, and the speed limit of r, respectively. $C(x.s, r.s)$ describes the activity confidence, which increases with the gap between $x.s$ and $r.s$. The confidence measure is used to ensure the reliability of composite activities as well as offering users a way to tune the system.

An ERM has desirable properties, described next:

- **Hierarchical structure**: a complex activity composed of many sub-actions can be represented by a "component parts of an object" relationship (Section. 7.2.5.2).

- **Multiple actors**: the mathematical foundation of ERM is set theory, so that an activity involving multiple actors can be defined naturally (Section. 7.2.5.3).

- **Context information**: low level observations can be represented by uncertain numerical values while background knowledge often needs to be described by symbolic representations. Observation and prior knowledge can easily be integrated (Section. 7.2.5.4).

7.2.5 Activity Inference

The ERM-based representation implies that inferring an activity is a search problem to find a subset of tracklets from entire data set, which satisfies certain conditions. ERM is implemented as a standard RDBMS and we can express set operations by SQL to find an activity from our database. The activity recognition problem is then equivalent to sending queries to the RDBMS.

A basic SQL statement has *SELECT*, *FROM*, and *WHERE* clauses: The *SELECT* command specifies the output attributes of entities, *FROM* defines the domain entities associated with the activity, and *WHERE* describes the set of relationships to define the activity and also its confidence. Activity definitions can easily be expressed by SQL statements.

There are some factors to classify activities in our domain: simple activities as simple motion patterns, sequence of activities as composite activities, multiple actor activities, and geospatial activities. In the following section we explain how to define different types of activities using ERM and how to infer those activities using SQL commands.

7.2.5.1 Example I: Simple Activity

Activities associated with motion patterns, such as *"U-turn"*, *"Loop"*, and *"3-point-turn"*, are easily defined and inferred by the ERM framework and its corresponding SQL statements.

Definition. A *"Loop"* is defined as a segmented track where there exists two tracklets $\{x_i, x_j\}$ whose Euclidean distance $\|x_i - x_j\|$ is smaller than the traveling distance: $Loop = \{x_i, x_j | (1 - \frac{\|x_i - x_j\|}{x_j.acc - x_i.acc}) > \theta, i < j, x_i.ID = x_j.ID\}$, where $(x_j.acc - x_i.acc)$ represent the traveling distance between x_i and x_j. The traveling distance is computed as the difference of the accumulated distances between these two tracklets.

The above definition is represented by SQL as shown in Table 7.2, where RDBMS tables T1, T2, and T3 come from the input tracklet table (e.g *SELECT* * INTO T1 *FROM* tracklet) and dist$(\cdot, \cdot)$ is a user defined function to compute the Euclidean distance.

```
SELECT * FROM T1, T2
WHERE
        T1.track_id = T2.track_id AND
        (1 - (dist(T1, T2)/(T2.acc - T1.acc))) > θ
```

Table 7.2: SQL: *"Loop"*

Figure 7.3 shows a result of *"Loop"* recognition, where a track contains several loops. Its corresponding SQL definition provides a set of tracklets associated with the activity.

7.2.5.2 Example II: Composite Activity

Suppose that we have three independent events identified as three entity sets: *"Entry"* (a_{En}), *"Stay"* (a_{St}) and *"Exit"* (a_{Ex}). *"Visit"* is a composite activity that can be described as a combination of these events.

Figure 7.3: An example of "loop"

Definition. We define *"Visit"* as the sequence of a_{En}, a_{St}, and a_{Ex}: $visit = \{x_j | i = j-1, k = j+1, x_i \in a_{En}, x_j \in a_{St}, x_k \in a_{Ex}, C(x_i)_{En}C(x_j)_{St}C(x_k)_{Ex} > \theta\}$, with x_i, x_j, x_k, three tracklets from the same track. This definition can be represented by SQL as shown in Table 7.3. Each identified event is a table, $Activity(ID, Confidence)$, that contains the IDs of tracklets and the confidence values. The confidence of *"Visit"* is defined as $C_{En}(x_i)C_{St}(x_j)C_{Ex}(x_k)$.

```
SELECT * FROM T1, T2, T3, En, St, Ex
WHERE
        T1.track_id = T2.track_id AND
        T2.track_id = T3.track_id AND
        T1.id + 1 = T2.id AND
        T2.id + 1 = T3.id AND
        T1.id = En.id AND
        T2.id = St.id AND
        T3.id = Ex.id AND
        (En.conf * St.conf * Ex.conf) > θ
```

Table 7.3: SQL: "*Visit*"

7.2.5.3 Example III: Multiple Actors Activity

Activities associated with multiple actors, such as *"Source"*, *"Sink"*, *"Convoy"*, and *"Following"*, can also be defined and inferred by ERM and SQL statements. We identify a source of tracks by finding a set of tracks that have the same starting location in different time periods.

Definition. Let us first define *"2-Source"* as a temporary set of 2 tracklets which exit from the same location: $2src = \{(x_i, x_j) | x_i.trackID \neq x_j.trackID, x_i \in a_{Ex}, x_j \in a_{Ex}, \|x_i - x_j\| < \omega\}$, where ω is a threshold. It provides a set of tracklet pairs which appear as many times as they are involved in a 2-tuple source. Extracting N-tuple source needs to count the number of occurrence for each tracklet: $source = \{x_i | S_i = \{(x_i, \cdot) \in 2src\}, |S_i| > \theta\}$, where $|S_i|$ is the cardinality of each subset S_i which contains the same tracklets in the pairs of the 2src set. Table 7.4 shows the corresponding SQL statements, where "Exit" action is represented as a table, Ex(id, confidence), that contains the IDs of tracklets and its confidence values.

Figure 7.1 shows a result of "SOURCE" recognition, where the location of results corresponds a parking lot.

```
SELECT T1.id as id INTO tmp FROM T1, T2, Ex
WHERE
        T1.track_id ≠ T2.track_id AND
        T1.id = Ex.id AND
        T2.id = Ex.id AND
        dist(T1, T2) > θ;
SELECT id, count(id) as confidence FROM tmp
GROUP BY id
```

Table 7.4: SQL: "*Source*"

7.2.5.4 Example IV: Geospatial Activities

The ERM framework is ideally suited to incorporate GIS information, as for *"Speeding"* (Section 7.2.4). Many activities can only be inferred within the context of geospatial information. We can find "all tracklets on a specific road" by looking at the correspondences between the locations of tracklets and the locations of known road segments.

Definition. *"On-road-X"* is a set of tracklets which are on the same road: $on\_road\_X = \{x | x.roadID = r.ID, r.name = "X", 1/\|x - r\| > \theta \, \forall r \in \mathcal{G}_{road}\}$, where r and $r.name$ designate a road segment and its name, and $\|x - r\|$ the Euclidian distance between the road segment and the tracklet. This definition is represented by SQL as shown in Table 7.5, where $T1$ and *Road* are the tracklet and road segment tables, respectively. Here, we compute the location of each tracklet in advance, and store the id of road segment into the tracklet table. The optional condition (1/dist(T1, Road) $> \theta$) provides a confidence measure.

```
SELECT * FROM T1, Road
WHERE
        T1.id = Road.id AND
        Road.name = "X" AND
        1/dist(T1, Road) > θ
```

Table 7.5: SQL: "*On-road-X*"

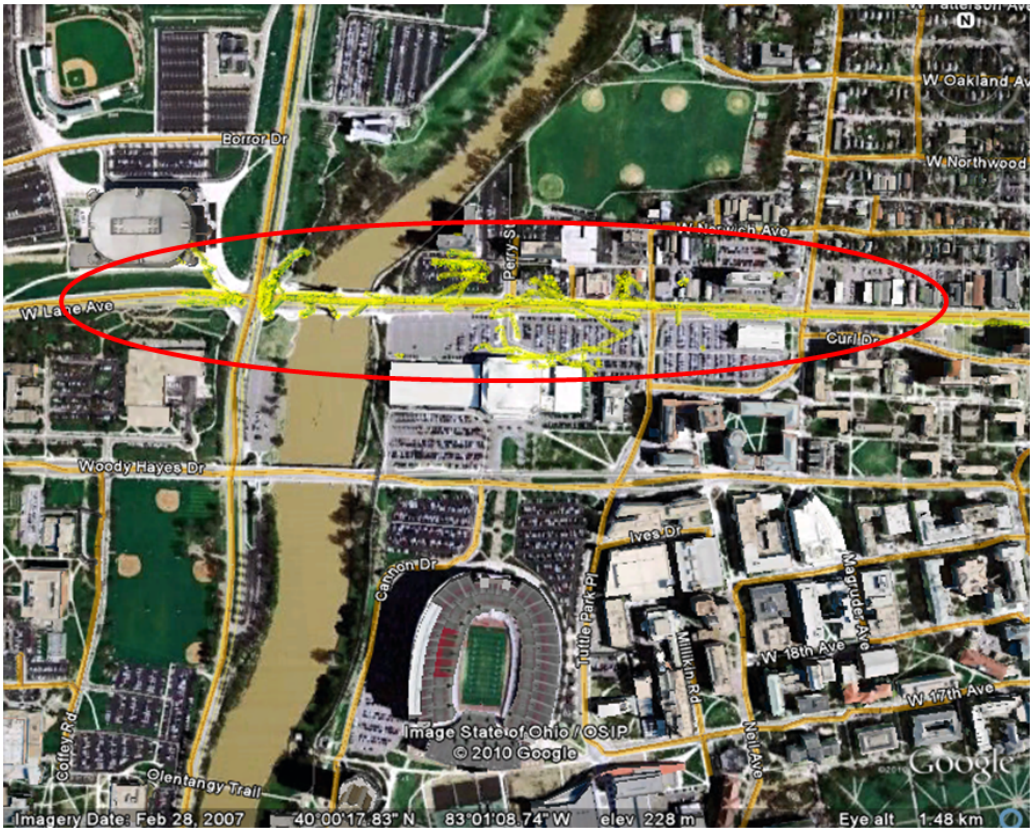


Figure 7.4: Orthographic view of "on road W Lane Ave". Geospatial data allows us to extract vehicles driving on a known road.

Figure 7.4 shows a result of *"On-road-X"* activity. Note that most spatial activities can also be enriched by having a geospatial attribute. For instance, a *"convoy"* becomes a "convoy traveling on highway X" when the spatial tracks are associated with geospatial information.

7.3 Results

We have implemented our framework using a standard RDBMS (MS-SQL [60]), and validated the approach on real visual tracks and GPS datasets. We define 7 activities for evaluation (including *"Loop"*):

- a three point turn (*"3PT"*) consists of two neighbor turns

  $\{x_i, x_j | ((x_i.\phi/\pi)(x_j.\phi/\pi))/(\|x_i - x_j\|) > \theta\}$

- a two point turn (*"2PT"*) has an acute angle $\{x | x.\phi > \theta\}$

- *"Stay"* is defined by the ratio between the time and travel distance between two points

  $\{x_i | (\|(x_j.time - x_i.time)\|)/(\|x_j.acc - x_j.acc\|) > \theta\}$

- *"U-turn"* has an acute angle turn between two tracklets which are located in the same road $\{x_j | (\|x_j.\phi < \pi/4, \|x_i - x_k\| < \omega_1, (x_k.acc - x_i.acc) > \omega_2\}$;

- *"Entry"* and *"Exit"* is defined with a stop, turns, speed changes and the travel distance. *"Entry"* is defined as $\{x_k | x_k.end = True, x_k.s < x_i.s, x_j.\phi > \omega_1, x_k.acc > \omega_2\}$.

$\{x_i, x_j, x_k\}$, $x.\phi$, and ω_i represent different tracklets from the same track, such as $i < j < k$, the turn angle attribute, and internal thresholds associated with the definition, respectively.

7.3.1 Real (CLIF) dataset

Data. The dataset is a set of tracking results extracted from the CLIF 2006 dataset [50]. This dataset contains wide area motion imagery captured from an airborne sensor. The sensor is

Figure 7.5: An example of extracted tracklets from the CLIF 2006 dataset [50]

composed of a matrix of 6 cameras, where the size of each image tile is 4008x2672. The video is captured at roughly 2 Hz, and it is in grayscale. The sequence is an example of persistent surveillance imagery, where the airplane makes several circular flyovers around the campus of Ohio State University. The footprint of the area where we computed tracks is about 1 km^2, and its duration is about 8 minutes. Each track is on average 1 minute long. The total number of tracks estimated in the sequence of interest is more than 8000.

The main challenge is the sheer number of objects, or points of interest, one must consider to determine whether an activity is happening. Furthermore, an activity is not a static concept

Out Actual	Loop	2PT	3PT	Entry	Exit	None
Loop	2	0	0	2	0	0
2PT	0	2	1	1	1	0
3PT	0	0	2	0	0	0
Entry	1	0	0	7	1	0
Exit	0	0	0	1	6	0
None	0	0	0	0	1	-

Table 7.6: Confusion Matrix (Real Data Set)

that one can identify at a glance. Instead, one must verify that a whole sequence of actions is happening to label something a particular activity.

Method. Our input is a set of tracks extracted by the tracking module described in Section. 7.2.3. Figure 7.5 shows a set of extracted tracklets. To build a set of ground truth data, from a set of automatically extracted tracks, we manually selected individual tracks that include pre-defined activities and assign labels for each data.

In our dataset, we used 2 loops, two 2 point turns, three 3 point turns, 8 entry and 7 exits. We inserted all selected tracks into a single table in our RDBMS and infer activities using pre-defined SQL statements.

Some tracks have more than one activity (e.g. a loop and a 3 point turn) but the locations associated with specific activities can be different. To evaluate the result of an activity, we extracted all tracklets, compared to the activity definition, from entire dataset, visualize the result using a GIS system (Google Earth), and then, verify manually whether the extracted tracklets represent the actual activity or not.

Results. Table 7.6 shows the confusion matrix among 5 activities, where "None" is a NULL activity to count missing and false alarms. Result shows that we can identify all simple

Out Actual	Loop	3PT	U-turn	Stay	None
Loop	17	0	0	0	0
3PT	0	7	0	0	0
U-turn	0	0	13	0	1
Stay	0	0	0	3	0
None	0	2	2	0	-

Table 7.7: Confusion Matrix (GPS Data Set)

Figure 7.6: An example of GPS tracks: (Left) A GPS track (yellow color) and a set of identified "Loops" (red color). (Right) One of the results shows a big closed circle and a small closed trajectory.

activities, such as "2 point turn", "3 point turn","Entry", "Exit", and "loop", which can easily seen in real data set.

In addition, we identified a number of geospatial activities, such as "on road X", "speeding", and "approaching X", as well as some complex activities including multiple actors, such as "source around X" and "sink around X". The extracted activities and geospatial objects can be visualized using an open GIS (e.g. Google Earth), where we can identify both activities and associated geospatial objects. Figure 7.4 in Section. 7.2.5.4 shows one of identified geospatial activities in the real dataset.

7.3.2 GPS trajectory dataset

Data. Manually labeling vehicle activity to extract a consistent ground truth from video tracks is a laborious work. Thus we also evaluated our activity recognition prototype on labeled data from GPS acquisition. Compared to the results we manually labeled from our tracking module, GPS tracks do not differ a lot. First the localization is mostly the same in both systems, with a 5 meters accuracy for the video geo-registration against 1 to 10 meters at 95% for our standard GPS. Second, the GPS acquisition frequency (1Hz) is only half our video framerate (2Hz).

The data acquisition consisted of daily trips between 10 and 40 minutes long. GPS filters were deactivated, so only raw data have been recorded. Finally, we manually identified every activity we planned to evaluate.

Method. We use the same tracking module to extract tracklets from the GPS dataset. To build a set of ground truth data, from a set of automatically extracted tracks, we manually selected individual tracks that include pre-defined activities and assign labels for each data.

The dataset includes 17 loops, 7 three point turns, and 13 u-turns. Since GPS tracks are much longer than the real tracks and each tracks include many activities, we inserted a single track into a single table in our RDBMS and inferred activities using pre-defined SQL statements.

To evaluate the result of an activity, we extracted a set of tracklets, corresponding to the activity definition, from a single track, visualize the result using a GIS (Google Earth), and then, verify manually both missing and false alarms.

Results. Table 7.7 shows the confusion matrix among 4 activities. Result shows that we can identify all simple activities, such as "Loop", "3 point turn", U-turn", and "Stay", which can easily seen in real data set. Fig. 7.6 shows a set of identified activities ("Loop") in a single GPS track.

7.4 Conclusion

Our results show that we can identify simple activities, such as "U-turn", "2 point turn", "3 point turn","Entry-Exit", "loop", and "speeding", as well as some complex activities including multiple actors, such as "source" and "sink". Extracted activities are visualized using an open GIS (e.g. Google Earth), where we can identify associated geospatial objects.

One limitation of this work is that all activities need to be hard-coded in advance by an operator. The automatic generation of activity descriptions from one or more observed instances, "query by example", will be an important future work, which might be equivalent to discover of hidden relationships between data entities.

Our system provides wide applications such as traffic monitoring, security, disaster control, public safety and law enforcement operations.

References

[1] S. Ali, V. Reilly, and M. Shah. Motion and appearance contexts for tracking and re-acquiring targets in aerial videos. In *IEEE CVPR*, pages 1--6, 2007.

[2] M. Andriluka, S. Roth, and B. Schiele. People-tracking-by-detection and people-detection-by-tracking. In *IEEE CVPR*, pages 1--8, 2008.

[3] A. Andriyenko and K. Schindler. Globally optimal multi-target tracking on a hexagonal lattice. In *ECCV*, volume 6311 of *LNCS*, pages 466--479. 2010.

[4] S. Avidan. Ensemble tracking. *IEEE PAMI*, 29(2):261--271, 2007.

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

[6] S. Baker, R. Gross, I. Matthews, and T. Ishikawa. Lucas-kanade 20 years on: A unifying framework: Part 2. Technical Report CMU-RI-TR-03-01, Robotics Institute, 2003.

[7] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Commun. ACM*, 4(6):284, 1961.

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

[9] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *ECCV*, volume 588 of *LNCS*, pages 237--252. Springer, 1992.

[10] A.F. Bobick and A.D. Wilson. A state-based approach to the representation and recognition of gesture. *IEEE PAMI*, 19(12):1325--1337, 1997.

[11] M. Brown and D.G. Lowe. Recognising panoramas. In *ICCV*, volume 2, pages 1218--1225, 2003.

[12] P. J. Burt and E. H. Adelson. A multiresolution spline with application to image mosaics. *ACM Trans. Graph.*, 2:217--236, 1983.

[13] A. A. Butt and R. T. Collins. Multiple target tracking using frame triplets. In *ACCV 2012*, volume 7726 of *LNCS*, pages 163--176. 2013.

[14] P. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9--36, 1976.

[15] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377--387, 1970.

[16] OpenStreetMap Contributors. Openstreetmap. `http://www.openstreetmap.org`.

[17] I.J. Cox and S.L. Hingorani. An efficient implementation of reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE PAMI*, 18(2):138--150, 1996.

[18] D. Damen. Activity analysis: Finding explanations for sets of events. *PhD Thesis. University of Leeds*, 2009.

[19] D. Damen and D. Hogg. Recognizing linked events: Searching the space of feasible explanations. In *IEEE CVPR*, pages 927--934, 2009.

[20] T.J. Darrell, I.A. Essa, and A.P. Pentland. Task-specific gesture analysis in real-time using interpolated views. *IEEE PAMI*, 18(12):1236--1242, 1996.

[21] R. David and H. Alla. Petri nets for modeling of dynamic systems: A survey. *Automatica*, 30(2):175--202, 1994.

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

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

[24] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381--395, 1981.

[25] A. R. J. François, R. Nevatia, J. R. Hobbs, and R. C. Bolles. Verl: An ontology framework for representing and annotating video events. *IEEE MultiMedia*, 12(4):76--86, 2005.

[26] F. Fung, K. Laskey, M. Pool, and M. Takikawa. Plasma: combining predicate logic and probability for information fusion and decision support. paper presented at the aaai spring symposium, 2005.

[27] Google. Google Earth. `http://www.google.com/earth/index.html`.

[28] H. Grabner, J. Matas, L. Van Gool, and P. Cattin. Tracking the invisible: Learning where the object might be. In *IEEE CVPR*, pages 1285--1292, 2010.

[29] D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society*, 51(2):271--279, 1989.

[30] R. Hamid, S. Maddi, A. Bobick, and M. Essa. Structure from statistics - unsupervised activity analysis using suffix trees. In *IEEE ICCV*, pages 1--8, 2007.

[31] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[32] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2003.

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

[34] C. Huang, B. Wu, and R. Nevatia. Robust object tracking by hierarchical association of detection responses. In *ECCV*, pages 788--801, 2008.

[35] T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24:1134--1141, July 2005.

[36] M. Irani, P. Anandan, and S. Hsu. Mosaic based representations of video sequences and their applications. In *ICCV*, pages 605--611, 1995.

[37] M. Irani, S. Hsu, and P. Anandan. Video compression using mosaic representations. *Signal Processing: Image Communication*, 7(4-6):529--552, 1995. Coding Techniques for Very Low Bit-Rate Video.

[38] Y. A. Ivanov and A. F. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE PAMI*, 22(8):852--872, 2000.

[39] H. Izadinia, V. Ramakrishna, K.M. Kitani, and D. Huber. Multi-pose multi-target tracking for activity understanding. In *IEEE WACV*, pages 385--390, 2013.

[40] S. Lao J. Xing, Haizhou A. Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses. In *IEEE CVPR*, pages 1200--1207, 2009.

[41] H. Jiang, S. Fels, and J.J. Little. A linear programming approach for multiple object tracking. In *IEEE CVPR*, pages 1--8, 2007.

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

[43] E.-Y. Kang, I. Cohen, and G. Medioni. A graph-based global registration for 2d mosaics. In *ICPR*, volume 1, pages 257--260, 2000.

[44] R. Kaucic, A.G. Amitha Perera, G. Brooksby, J. Kaufhold, and A. Hoogs. A unified framework for tracking through occlusions and across sensor gaps. In *IEEE CVPR*, volume 1, pages 990--997, 2005.

[45] 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.

[46] Z. Khan, T. Balch, and F. Dellaert. Multitarget tracking with split and merged measurements. In *IEEE CVPR*, volume 1, pages 605--610, 2005.

[47] S. J. Kim and M. Pollefeys. Radiometric alignment of image sequences. In *IEEE CVPR*, volume 1, pages 645--651, 2004.

[48] J. Kleinberg and E. Tardos. *Algorithm Design*. Pearson, 2006.

[49] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498--519, 2001.

[50] Air Force Research Lab. CLIF 2006. `https://www.sdms.afrl.af.mil/index.php?collection=clif2006`.

[51] Air Force Research Lab. WPAFB-21Oct2009. `https://www.sdms.afrl.af.mil/`.

[52] A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless image stitching in the gradient domain. In *ECCV*, volume 3024 of *LNCS*, pages 377--389. Springer, 2004.

[53] 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.

[54] W.-Y. Lin, S. Liu, Y. Matsushita, T.-T. Ng, and L.-F. Cheong. Smoothly varying affine stitching. In *IEEE CVPR*, pages 345--352, 2011.

[55] F. Liu, M. Gleicher, H. Jin, and A. Agarwala. Content-preserving warps for 3d video stabilization. *ACM Trans. Graph.*, 28:44:1--44:9, July 2009.

[56] M.I.A. Lourakis. levmar: Levenberg-marquardt nonlinear least squares algorithms in C/C++. `http://www.ics.forth.gr/~lourakis/levmar/`, 2009.

[57] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91--110, 2004.

[58] S. Mann and R.W. Picard. Virtual bellows: constructing high quality stills from video. In *IEEE ICIP*, volume 1, pages 363--367, 1994.

[59] G. Medioni, I. Cohen, F. Bremond, S. Hongeng, and R. Nevatia. Event detection and analysis from video streams. *IEEE PAMI*, 23(8):873--889, 2001.

[60] Microsoft. Microsoft sql server. `http://www.microsoft.com/sqlserver/`.

[61] P. J. Moreno, P. P. Ho, and N. Vasconcelos. A kullback-leibler divergence based kernel for svm classification in multimedia applications. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

[62] K. Okuma, A. Taleghani, N. de Freitas, J. Little, and D. Lowe. A boosted particle filter: Multitarget detection and tracking. In *ECCV*, volume 3021 of *LNCS*, pages 28--39, 2004.

[63] A.G.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.

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

[65] E. Pollard, B. Pannetier, and M. Rombaut. Convoy detection processing by using the hybrid algorithm (gmcphd/vs-immc-mht) and dynamic bayesian networks. In *Information Fusion, 2009. FUSION '09. 12th International Conference on*, pages 907--914, 2009.

[66] A. B. Poore. Multidimensional assignment formulation of data association problems arising from multitarget and multisensor tracking. *Computational Optimization and Applications*, 3:27--57, 1994.

[67] J. Prokaj, M. Duchaineau, and G. Medioni. Inferring tracklets for multi-object tracking. In *IEEE CVPRW (WAVP)*, pages 37--44, 2011.

[68] J. Prokaj and G. Medioni. Accurate efficient mosaicking for wide area aerial surveillance. In *IEEE WACV*, pages 273--280, 2012.

[69] D. Reid. An algorithm for tracking multiple targets. *Automatic Control, IEEE Transactions on*, 24(6):843--854, Dec 1979.

[70] 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. Springer, 2010.

[71] R. L. Rothrock and O. E. Drummond. Performance metrics for multiple-sensor multiple-target tracking. In *Proceedings of SPIE*, volume 4048, pages 521--531, 2000.

[72] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43--49, February 1978.

[73] D. Salvi, J. Waggoner, A. Temlyakov, and S. Wang. A graph-based algorithm for multi-target tracking with occlusion. In *IEEE WACV*, pages 489--496, 2013.

[74] H. Sawhney, S. Hsu, and R. Kumar. Robust video mosaicing through topology inference and local to global alignment. In *ECCV*, volume 1407 of *LNCS*, pages 103--119. Springer, 1998.

[75] H.S. Sawhney and R. Kumar. True multi-image alignment and its application to mosaicing and lens distortion correction. *IEEE PAMI*, 21(3):235--243, 1999.

[76] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7--42, 2002.

[77] G. Schweighofer and A. Pinz. Robust pose estimation from a planar target. *IEEE PAMI*, 28(12):2024--2030, 2006.

[78] 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.

[79] H.-Y. Shum and R. Szeliski. Systems and experiment paper: Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision*, 36:101--130, 2000.

[80] S. N. Sinha and M. Pollefeys. Pan-tilt-zoom camera calibration and high-resolution mosaic generation. *Computer Vision and Image Understanding*, 103(3):170--183, 2006. Special issue on Omnidirectional Vision and Camera Networks.

[81] D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss. Tightening lp relaxations for map using message passing. 2008.

[82] C. Stauffer. Estimating tracking sources and sinks. In *IEEE CVPRW*, volume 4, page 35, 2003.

[83] U.S. Geological Survey. USGS EarthExplorer. `http://edcsns17.cr.usgs.gov/NewEarthExplorer/`.

[84] R. Szeliski. Image mosaicing for tele-reality applications. In *IEEE WACV*, pages 44--53, dec 1994.

[85] R. Szeliski. Image alignment and stitching: A tutorial. Technical Report MSR-TR-2004-92, Microsoft Research, 2006.

[86] E. Tola, V. Lepetit, and P. Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE PAMI*, 32(5):815--830, 2010.

[87] P. Turaga, R. Chellappa, V.S. Subrahmanian, and O. Udrea. Machine recognition of human activities: A survey. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(11):1473--1488, 2008.

[88] 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.

[89] J. Xiao, H. Cheng, F. Han, and H. Sawhney. Geo-spatial aerial video processing for scene understanding and object tracking. In *IEEE CVPR*, pages 1--8, 2008.

[90] 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.

[91] M. Yang, Y. Wu, and G. Hua. Context-aware visual tracking. *IEEE PAMI*, 31(7):1195--1209, 2009.

[92] Q. Yu and G. Medioni. Multiple-target tracking by spatiotemporal monte carlo markov chain data association. *IEEE PAMI*, 31(12):2196--2210, 2009.

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