VICTORIOUS: Video Indexing with Combined Tracking and Object Recognition for Improved Object Understanding in Scenes

by

Yuetian Xu
S.B., Massachusetts Institute of Technology (2008)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2009

Author .....
Department of Electrical Engineering and Computer Science
August 11, 2009

Certified .....
Richard W. Madison
Senior Member of the Technical Staff, C.S. Draper Laboratory
Thesis Supervisor

Certified by .....
Tomaso A. Poggio
Professor, Computer Science and Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by .....
Christopher J. Terman
Chairman, Department Committee on Graduate Theses

VICTORIOUS: Video Indexing with Combined Tracking and Object Recognition for Improved Object Understanding in Scenes

by

Yuetian Xu

Submitted to the Department of Electrical Engineering and Computer Science
on August 11, 2009, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Automatic understanding of video content is a problem which grows in importance every day. Video understanding algorithms require accuracy, robustness, speed, and scalability. Accuracy generates user confidence in usage. Robustness enables greater autonomy and reduced human intervention. Applications such as navigation and mapping demand real-time performance. Scalability is also important for maintaining high speed while expanding capacity to multiple users and sensors.

In this thesis, I propose a "bag-of-phrases" model to improve the accuracy and robustness of the popular "bag-of-words" models. This model applies a "geometric grammar" to add structural constraints to the unordered "bag-of-words." I incorporate this model into an architecture which combines an object recognizer, a tracker, and a geolocation module. This architecture has the ability to use the complementarity of its components to compensate for its weaknesses. This allows for improvements in accuracy, robustness, and speed.

Subsequently, I introduce VICTORIOUS, a fast implementation of the proposed architecture. Evaluation on computer-generated data as well as Caltech-101 indicate that this implementation is accurate, robust, and capable of performing in real time on current generation hardware.

This implementation, together with the "bag-of-phrases" model and integrated architecture, forms a step towards meeting the requirements for an accurate, robust, real-time vision system.

Thesis Supervisor: Richard W. Madison
Title: Senior Member of the Technical Staff, C.S. Draper Laboratory

Thesis Supervisor: Tomaso A. Poggio
Title: Professor, Computer Science and Artificial Intelligence Laboratory

Acknowledgments

I have been extremely fortunate in having Richard Madison as my advisor. Rich is committed to helping his students succeed. He is always available to offer support and guidance on technical issues and otherwise. His help in everything from feedback during brainstorming to writing and presentation has been invaluable to this work.

Prof. Tomaso Poggio is also a wonderful advisor. His extensive knowledge of the literature and insights into the human vision system have helped provide some theoretical grounding to my desire to just go out and build some implementations.

I also owe thanks to my colleagues at Draper Laboratory, many of whom I worked with on projects related to this thesis. In particular, Yechezkal Gutfreund authored the GUI system which I used for displaying my results; Caitrin Eaton and Adam Ray helped with some of the experimentation.

Finally, I would like to thank my parents and grandparents for encouraging curiosity and learning from an early age. They inspire me to strive for the best and not settle for anything less.

This thesis was prepared at the Charles Stark Draper Laboratory, Inc., under the Independent Research and Development Numbers 22920-002 and 23921-003, titled "Image Search By Content".

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Rapid understanding and indexing of large amounts of video data is a challenge that grows more important every day. A camera providing video imagery has the advantage of being a cheap and powerful sensor. As a result, there is increasing use of video for purposes of surveillance and mapping, among others. The ability to process that video in an automated and speedy way would greatly reduce demands on human operators. The ability to perform video understanding in real time would not only enable more data to be processed for the aforementioned applications but also enable other applications such as landmark-based navigation (whether for cars, planes, or UAVs) and interactive robots (which might include household robots like Sony's AIBO^{TM}or iRobot's Roomba^{TM}). There is great demand for fast and robust algorithms for video understanding and effective system architectures which combine these algorithms to produce practical implementations.

The design of a fast, robust system supporting many types of video understanding tasks presents many challenges. In this chapter, I first give a definition of video understanding and its components as well as some of the challenges associated with each component. I describe several applications utilizing some of these components and challenges faced when speeding up the processing. This is followed by an overview of my contributions to addressing these challenges.

Video understanding is the problem of taking a video as input and subsequently identifying objects (object recognition), establishing consistent identities of objects

over time (tracking), and finding the location of the object in relative and/or absolute terms (geolocation). These three components together or in part can provide varying degrees of scene understanding. In particular, an implementation which combines these three components can be very useful for either autonomous robotic navigation or in assisting humans in performing video analysis among other applications.

In a system for doing this video understanding, the object recognizer needs to have both *instance* and *category* recognition capabilities. Some applications demand the ability to identify specific instances of object such as "Bob's car". Other applications demand the capability to show a user a list of all "cars" in the video which involves the generic category of "car". The former can perform things like landmark navigation while the latter allows the search space for the response to be tractable by not requiring computation for every possible instance. Instance recognition can also be useful when combined with tracking to establish consistent identity of specific objects from frame to frame. When additional telemetry information is available about the position and orientation of the camera, then an accurate geolocation estimate can also be formed.

1.1 Video Understanding Applications

A video understanding system would have many different applications such as category search on video, example-based search on video, user-specified object tracking, and landmark navigation. These applications highlight the need for several design requirements.

1. **Accuracy:** a high degree of accuracy is necessary to provide confidence in a user-facing system as well as reasonable performance in an automated system.

2. **Robustness:** good robustness is necessary to maintain the desired level of accuracy across challenging operating conditions containing image noise, background clutter, illumination variations, *etc.*

3. **Speed:** whether for a completely automated system or for a user, speed is

paramount. In fact, for navigation purposes, the system will be unusable if it lacks sufficient speed.

4. **Scalability:** potentially many sensors can contribute information and many users access the information simultaneously.

In following sections, I present some examples of usage cases of each application and their demands on the requirements listed above. In all of the cases, a high level of accuracy is necessary for successful application of a video understanding system.

Category Search on Video

Category search on video enables a human operator to perform keyword search on a set of video that is previously processed with a set of learned categories. This requires speed as even if the video is previous processed, the preprocessing should preferably be quick. Furthermore, sufficiently fast speed may allow for the user to upload novel video for processing. Robustness will improve user experience so that the list of results shown to the user is not excessively long. Note that the user may not be sensitive to the exact ranking as long as the correct results are near the top. Scalability is also a concern as ideally the system should be able to handle multiple users simultaneously and also allow for cross referencing the different videos automatically to improve accuracy.

Example-based Search on Video

Example-based search on video allows a human operator to provide the system with an example of an object that should be searched for. This can take the form of a set of pictures showing the object or possibly a CAD model. The system will take this model and then search through the available video for instances of this object or similar objects. In this case, speed is paramount as the entire space of videos must be searched for this object. The time this takes directly affects the response time for the user. Robustness is important. Although again, a little leeway among the top few

results is possible with user involvement. Scalability is a major concern in searching over a large set of videos.

User-specified Object Tracking

User-specified object tracking enables the user to select an area of a frame of video representing an object in that frame and ask the system to track said object across subsequent frames. Some additional information, such as what type of object it is, may be given. Speed is again important as we would like to be able to show real-time tracking to the user. Furthermore, robustness is still important. While some adjustments by the user might be possible to account for tracking errors, demand for frequent adjustments might greatly inconvenience a user and offer a poor experience. Scalability is again a concern as we would like to support tracking multiple such objects simultaneously in a video or even multiple videos (with accurate geolocation, the user only needs to select the object in a single video but would be able to track it across many other videos).

Landmark Navigation

Landmark navigation allows an autonomous vehicle/robot to refine its own geolocation estimate by examining known, visible landmarks and localizing relative to those landmarks. This can be useful in areas where Global Positioning System (GPS) is unavailable and can still allow for correction to Inertial Measurement Unit (IMU) drift. Speed is highly important in this case as navigation tasks need to occur in real time to prevent collisions and other undesirable consequences. Robustness is also highly important to prevent false matches with landmarks which can greatly throw off position estimates. Lastly, scalability is also a concern due to the number of landmarks which might be used to provide better navigation estimates.

1.2 Problems with existing vision algorithms

Many computer vision algorithms today tackle the problem of video understanding in terms of its separate component tasks and aim to build separate object classifiers, trackers, and geolocation components. While this separation is tempting from a systems view to produce modularity, there is much to be gained by interlocking these components and exploiting their mutual information. Such a holistic design would yield better performance than ones which maintained separate components.

1.3 Thesis Overview

1.3.1 Contributions

In this thesis, I develop a framework architecture that efficiently combines an object classifier, a feature tracker, and a geolocation module into the same system. This architecture utilizes a client-server model and allows for scalability to enable multiple users, multi-sensor fusion, and ease of parallelization. Unlike previous systems, this architecture efficiently utilizes interlocking constraints from its components to achieve gains in both accuracy and speed on several video understanding tasks.

For the object classifier, I describe a novel approach of extending a "bag-of-words" model to a "bag-of-phrases" model which retains much of the speed benefits of "bag-of-words" while greatly improving robustness in the face of noise and clutter. I also describe some ways to speed up the classifier at the cost of sacrificing very little accuracy by utilizing priors from a tracker as well as porting select code over to a Graphic Processing Unit (GPU).

For the tracker, I describe a method of utilizing knowledge about object orientation to improve tracking performance by using better templates. Similarly, a robust object classifier and tracker together produce successive lines of sight which are able to produce a good geolocation estimate.

My research has led me to develop VICTORIOUS, a fast and efficient implementation of the architecture described above. This implementation is demonstrated to

be superior to some existing solutions on several video understanding tasks.

1.3.2 Outline

The next chapter gives a description of some current object classifiers and trackers, particularly with some information on their performance with regards to robustness and scalability. Chapter 3 describes an object classifier system based on a "bag-of-phrases" model and demonstrates improvement compared with a "bag-of-words" baseline. Chapter 4 presents a proposed system architecture demonstrating interactions between a tracker, a geolocation module, and an object classifier. Chapter 5 shows some implementation details of VICTORIOUS and experimental results on the performance of VICTORIOUS. Chapter 6 offers some concluding remarks and recommendations for future work.

Chapter 2

Background and Related Work

This chapter will discuss some of the related work in the field of video understanding. This includes the components of object recognition (Section 2.1), tracking (Section 2.2), and geolocation (Section 2.3).

2.1 Object Recognition

Object recognition is a rich field in which significant work has been done. This section discusses briefly a small subset of the field most relevant to this thesis. For a good survey of the field, see the ICCV short course from Fei-Fei *et al.* [14]. Another excellent treatment can be found in Rich Szeliski's upcoming book [40].

On the broadest scale, object recognition can be divided into *instance* recognition and *category* recognition. Instance recognition is the problem of recognizing specific objects. One example would be the question "Is that *my* car?" Category recognition, on the other hand, is the problem of recognizing whether something belongs to a category. An example of this would be the question "Is that *a* car?"

2.1.1 Challenges

In both instance recognition and category recognition, many challenges exist. Some of these include illumination variations, scale, rotation (both in-plane and out-of-plane),

noise, occlusion, etc. These combine to make relatively simple approaches such as sum-of-squared-differences or normalized cross-correlation perform relatively poorly.

2.1.2 Bag-of-Words

One particular approach which is able to handle most of these challenges is a "bag-of-words" approach. This approach considers an object to be an unordered bag of features. The distribution of observed features can give indications as to what object exists in the scene. This approach has the benefit of being robust to things such as illumination variations, scale, rotation, and noise if the features extracted are robust. Also, some robustness to occlusion can be attained due to the fact that the non-occluded parts will still generate features. This approach overall offers a fairly good blend of speed and performance. Many of the top performing systems on benchmarks such as the Caltech-101 dataset [22] and the Pascal Visual Object Classes Challenge dataset [12] are based upon this method.

Robust, Discriminative Features

A vital part of "bag-of-words," or indeed any feature-based approach, is heavily dependent upon a good feature descriptor. A good feature descriptor needs to be both robust (invariant to nuisance parameters) and discriminative (capable of distinguishing between features with different underlying meaning). Designing a good feature descriptor involves a careful trade-off between these two properties. Either end of the spectrum is undesirable. In one extreme where even the most minuscule differences results in two features being considered different, there is perfect discrimination but no robustness. In the other extreme where every feature is considered the same, then there is perfect robustness but no discrimination.

Several feature descriptors developed in recent years produce the desired blend of robustness and discrimination. Lowe's SIFT descriptor [31] is a particularly noteworthy and successful example. SIFT is robust to scale and in-plane rotations by default. In practice, it provides fairly good robustness to illumination variations, out-

of-plane rotation, and noise. It produces a fairly good balance between robustness and discrimination.

Several other descriptors implement ideas similar to that of SIFT or extend it directly. Examples in this category include geometric histogram [3], shape context[5], Gradient Location and Orientation Histogram (GLOH) [34], PCA-SIFT[27], and Color SIFT[7].

Mikolajczyk and Schmid [34] evaluated several of the feature descriptors listed above as well as others. They concluded that SIFT and GLOH performed the best.

Document Retrieval and pLSA

The "bag-of-words" model of object recognition is derived from its resemblance to its namesake model of topic detection in document retrieval. Just as the presence of a topic in a document can be inferred by the distribution of the observed words, the presence of an object in an image can be inferred by the distribution of the features. Some of the earliest work to apply this document retrieval analogy to object recognition includes Csurka *et al.* [10] and Sivic and Zisserman [39].

One popular and mathematically principled way of doing "bag-of-words" involves probabilistic Latent Semantic Analysis (pLSA) first introduced by Hofmann [21]. PLSA is based upon mixture decomposition of a latent class model. PLSA for topic detection models the topic (z) as a latent variable from the observed variables of the document (d) and word distribution (w). Formally, pLSA attempts to decompose a co-occurrance matrix of word and document $P(w, d)$. Model densities $P(w|z)$ and $P(z|d)$ are learned using the Expectation Maximization (EM) algorithm. The document retrieval analogy as described in the previous paragraph can be used to apply pLSA to object recognition.

2.1.3 Ways to Improve Bag-of-Words

Pure "bag-of-words" type approaches do have limitations to their performance. For example, Figure 2-1 shows an example of two very different objects which would be

considered identical by "bag-of-words" for having the same parts. This failure can be largely attributed to a lack of spatial awareness. Some select methods for overcoming this lack of spatial awareness are presented below. These include spatially aware kernels, voting, spatial histograms, and geometric constraints.

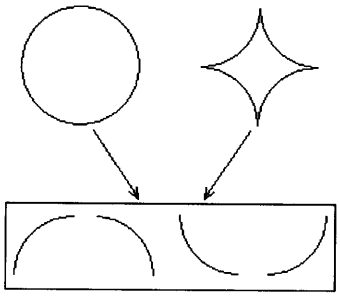

Figure 2-1: Bag-of-words has no spatial awareness and will thus consider two very different objects (left and right) identical due to both containing the same set of parts (boxed).

Spatially Aware Kernels

One method is to modify the feature descriptors to be spatially aware. Rather than capturing information only in a local area, the spatially aware descriptor captures information in a larger area. If this is done naively, then invariance will suffer significantly. The solution is to produce a blurring which increases with greater distance from the center. The geometric blur filter introduced in Berg *et al.* [6] is an example of a spatially aware kernel.

Voting

One very successful and well-engineered instance recognition system is Video Google by Sivic and Zisserman. In their paper [39], they demonstrated the benefits of filtering for spatial consistency on rejecting spurious matches. Their notion of spatial consistency required that a match between two features are supported by some other match between the neighbors of said features in both images. This enforced condition that legitimate matches co-occur was reported to produce good false positive rejection.

Spatial Histograms

Other work has also explored extending "bag-of-words" models to make them spatially aware such as ABSolute position pLSA(ABS-pLSA) and Translation and Scale Invariant pLSA (TSI-pLSA) introduced in Fergus *et al.* [15]. ABS-pLSA adds a location term x (quantized into bins) to the co-occurrence matrix being modeled in pLSA to produce $P(w, x, d)$. TSI-pLSA expands on this further by adding in position relative to centroid of a subwindow c to produce translation and scale invariance.

Lazebnik *et al.* [28] presented an approach using a spatial pyramid of histograms rather than a single histogram. Each level of this pyramid contains a collection of histograms of sub-windows of the image.

Geometric Constraints

Applying geometric constraints to improve object recognition accuracy is an idea that was previously explored in several papers ([17],[18],[36],[49]). Grimson and Lozano-Perez ([17],[18]) modeled objects as polyhedra and calculated several geometric constraints on distance and angles of faces and their normals. Schmid and Mohr [36] calculated angles between interest points and obtained results which show better separation between the top result and the rest. Zhang *et al.* [49] used relative distance as a constraint for matching and weighted results both by goodness of match and distance to the match. Much of this group of work was done more than a decade ago with fairly simple models or features and often in restricted domains *e.g.*, recognition of industrial tools in binary images.

2.2 Tracking

Some good surveys of tracking include Lepetit and Fua [29] and Yilmaz *et al.* [48]. The Lepetit and Fua survey is especially thorough on the topic of rigid object tracking and contains over 100 references. The field of tracking is very large and we will touch briefly on some of the most relevant topics.

Much of the earlier work on tracking was based around optical flow. Barron *et al.* [4] compared several methods of optical flow tracking including those of Anandan [2], Fleet-Jepson [16], Horn-Schunk [23], and Lucas-Kanade [32]. Barron *et al.* concluded that Lucas-Kanade and Fleet-Jepson methods performed the best.

The introduction of feature descriptors saw a shift away from patch-based tracking towards feature-based tracking. Some works which explored this include Ravela *et al.* [35] as well as Uenohara and Kanade [43].

Although originally formulated as an optical flow tracker, the Lucas-Kanade can in fact be formulated as a local interest point tracker as well. Hager and Belhumeur [19] showed a more effective formulation of Lucas-Kanade. The combination of speed and accuracy offered by Lucas-Kanade led to widespread adoption in applications such as 3D tracking [8, 25, 26].

One approach to 3D tracking has been focused on the common special case of planar objects. Part of the intuition is the fact that many man-made objects possess the quality of having surfaces which are roughly planar. Both the approaches of Jurie and Dhome [26] and Simon *et al.* [38] make this planar assumption and have been fairly successful.

Drifting off the object in question is always a challenge for tracking. Several approaches have addressed this via the use of key frames [35, 9, 11, 44]. Vacchetti and Lepetit's approach [44] serves as a good example. First, they pick a key frame closest in viewpoint to the last known one. They then produce an intermediate image from warping between the key frame and input frame. The tracking is then performed between the intermediate frame and the new frame. This approach works well for the most part. However, key frame approaches have the tendency to be wide-baseline and is thus less accurate than the short-baseline frame-to-frame approaches. A good example of a short-baseline frame-to-frame approach can be found in Zhang *et al.* [49]. In that paper, they track features by searching the surrounding region in the next frame.

2.3 Geolocation

A multitude of photogrammetry books contain a good treatment of the problem of localizing an object in space given lines-of-sight. One such book is authored by Hartley and Zisserman [20].

Of particular interest to this work are practical systems which demonstrate geolocation capabilities and serve as points of comparison. One such system which works on small Unmanned Aerial Vehicles (UAVs) was described in Madison *et al.* [33]. In that particular case, they were able to achieve a geolocation accuracy of 10 meters.

Chapter 3

Viewpoint Sensitive Classifier

3.1 Overview

"Bag-of-words" models have proven to be a fast and fairly accurate way to perform object recognition. However, they utilize only local features and do not capture the relative geometric relationships between said features. This chapter proposes a framework for constructing "bag-of-phrases" models for object recognition. This approach extends traditional "bag-of-words" models to considering bigrams of "visual words" as well as applying a "geometric grammar" which is learned from the local geometric relationships between the "visual words". This greatly increases the false-positive rejection rate. This approach also provides significant improvements in robustness to background clutter and noise compared with generic "bag-of-words" approaches. Two variants of this general approach are developed to demonstrate its benefits for both instance recognition and category recognition.

3.2 Motivation

3.2.1 Where Bag-of-Words Fails

"Bag-of-words" models treat an image as an independent set of parts each corresponding to parts of an object. These are dubbed "visual words" given their analog

in document retrieval. A histogram of features, each representing an image part, is created for any image under consideration. This histogram is compared against the histogram of a known object to assess whether that object shows up in the image. While this approach works well in many instances, there are limitations to its possible performance. These inherent and unsurmountable limitations derive from ignoring geometric context information. For example, consider the scenario in Figure 3-1 where an entire car and a random jumble of parts from the same car would produce identical histograms. However, humans would not consider the second image to contain a car.

Figure 3-1: A bag-of-words model is unable to distinguish between a car (left) and a jumble of car parts (right). Image of car on the left was found through a Google image search.

Arguments can be made that it's unlikely that such a scrambled car exists in real life and thus we need not worry about such scenarios. However, no feature detector and matcher is perfect in terms of robustness. Thus, scenarios where noise and/or clutter produce a histogram similar to that of a car is more likely than one might expect. The likelihood that such a false detection appears as a scrambled car is high while the likelihood that it resembles an ordered car is vanishingly low.

Another possible scenario occurs if the features being detected are in fact legitimate features from another object. Suppose the imaging conditions are sufficiently degraded that only some windows and wheels can be reliably detected. Based purely on a histogram, one might not know if the detected object is a car or a plane as both have windows and wheels. Thus, from purely local information about the features,

it's difficult to tell if they are from one class or another.

This problem can be partially overcome by producing better feature detectors. Making a feature detector invariably involves some trade-off between invariance and discriminative power. While improving feature detectors will move the curve out towards perfect discrimination and perfect invariance, it is very likely that such a point is unattainable. The limitations of the power of local discrimination is apparent even in human vision. Torralba *et al.* in [42] note that in low resolution imagery, it is often possible for humans to successfully segment a scene and identify objects even if the individual objects are very low resolution and cannot be identified by themselves. This suggests top down constraints provided by spatial information plays a role and that local information by themselves is not enough.

3.2.2 Adding Some Order to the Bag

Why Bag of Words is Salvageable

Despite failures such as the example shown in Figure 3-1, there is still reason for optimism and no need to abandon the image vocabulary approach of "bag-of-words" entirely. The feature descriptors typically used for the task such as SIFT still tend to have good *invariance* under challenging imaging conditions. However, they suffer from the problem of relatively poor *discrimination* as local information alone is insufficient for distinguishing between the true positive matches and false positives. The solution is to take into account *semi-local* information to improve the *discrimination.*

Hierarchical Histograms

One common proposal for incorporating *semi-local* information is to subdivide the image at successive levels and generate histograms for each subdivision. This creates a hierarchy of histograms that incorporates spatial information. Usually, higher scores are given for matching features at a finer subdivision level.

This may come at the expense of some invariance which the feature detectors possess by adding in assumptions about the nature of the target based upon the

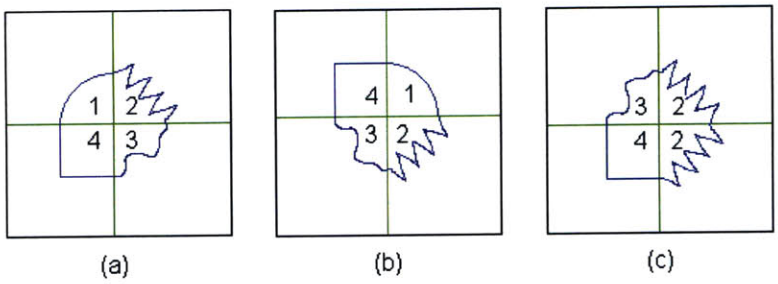

Figure 3-2: Failure of hierarchical spatial histograms to match objects correctly. In (a), the object being searched for is outlined in blue and the features are labeled in all four subdivisions (shown in green). (b) is a rotated version of (a) but has no matches at the subdivision level, only 4 matches at top level. On the other hand, (c) is different from (a) but has 2 matches at subdivision level and 3 matches at the top level. Since subdivision level matches are weighted more heavily than top level ones, (c) is considered a better match than (b) to (a).

geometry of the histograms used. For example, should a rectangular hierarchical histogram be used, even if the features are invariant to rotation, a rotation of an object in the image would result in a very different set of histograms. In Figure 3-2, we can see that even though the features are rotationally invariant, a different object can still score higher than a rotated version of the desired object.

Furthermore, we add dependence on histogram boundaries which may be unstable. This can be seen in examples where even slight translations and rotations can move one feature from one histogram to another. Again, this is assuming an environment with little or no false positives. Histograms can be notoriously sensitive to false matches. A way which addresses these weaknesses is "geometric grammars" which we formulate in the next section.

3.2.3 Geometric Grammar

Another way of adding *semi-local* information is to apply a "geometric grammar". Just as real words must fit certain grammatical rules to be considered a valid phrase, we can also enforce consistency according to a "geometric grammar" so as to validate a pair of "visual words" as forming a subassembly.

In natural language processing, fairly coherent sentences can be automatically

generated by using bigram and trigram (successive couplets and triplets of words) statistics. This demonstrates the power of *semi-local* information. In a similar vein, we can take pairs of "visual words" close to each other and compute geometric statistics with them which we can then use to check against our "visual grammar". There is much lower probability of false positives managing to not only match both fairly discriminative feature descriptors but also satisfy all the "visual grammar" rules which are applied to the individual "visual words".

The following sections will describe a "geometric grammar" developed for SIFT which was used in the system described. In addition, the properties of a good "geometric grammar" are quantified.

Specialized Grammar for SIFT

While the concept of "geometric grammar" is a general one which can be applied to any local feature detector, we will focus on application to SIFT features. SIFT is chosen due to its demonstrated performance in comparison with other local feature descriptors [34].

A SIFT feature not only consists of a feature descriptor but also information about its location in the image, its scale, and absolute orientation. While "bag-of-words" approach is concerned solely with the descriptor and discard the geometric information, there are several useful pairwise geometric measures from said information. These measures generated values to compare against which we form "grammar" rules from.

What Makes a Good Grammar?

The design goal of a good "geometric grammar" is to maintain the *invariance* of the feature detector while improving *discrimination* as much as possible.

A "geometric grammar" consists of a set of rules, each based upon the values of a particular "geometric measure". In light of the stated design goals, these geometric measures must satisfy two important properties. The first property (denoted as the invariance property) states each geometric measure must preserve all the invariance of

the feature descriptors (scale and in-plane rotation in the case of SIFT). The second property (denoted as the discrimination property) states the set of all the measures must be sufficient to fully specify the geometric properties of the descriptors given their relative positions.

The first property satisfies the first design goal. The second property indicates that from an information theory point of view, every last bit of information regarding the relative geometric relationship of the two features is extracted. In other words, the set of geometric measures forms a spanning set of sorts over the geometric information. From a computational point of view, the set should preferably be a minimal spanning set. This can be taken as an optional design goal if desired. In either case, the two stated properties are sufficient to satisfy our stated design goal of maximizing *discrimination* without sacrificing *invariance*.

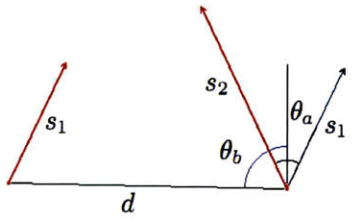

Figure 3-3: Illustration of two SIFT features and some associated geometric measures. The SIFT features are the red vectors labeled s_1 and s_2. d is the length of the connecting line segment. θ_a is the signed difference in orientation. θ_b is the signed difference between the average orientation and the connecting line segment.

3.2.4 Proposed Grammar

A proposed set of four geometric measures which encapsulates the relative geometric information between the pair of SIFT features illustrated in Figure 3-3 is listed below.

1. Ratio of scales of the features: $\frac{s_1}{s_2}$.
2. Ratio of the average scale of the features to the distance of the connecting line segment: $\frac{s_1+s_2}{2d}$.
3. Signed difference in orientation of the features: θ_a.

4. Signed difference between the average orientation of the features and the connecting line segment: θ_b.

Due to the fact that the pairs of scale and orientation measures are all calculated in relative and not absolute terms, they are invariant to scale and in-plane rotation. The measures thus satisfy the desired invariance property.

Given the relative positions, a connecting line segment can be drawn between them. Combining this with the fourth measure gives us the average orientation of the features. The average orientation in conjunction with third measure fully determine the direction of both features. Meanwhile, the distance between the two features and the second measure yields the average scale of the features. The first measure can then be applied to produce the scale of both features. Therefore, the set of four measures fully determine the scale and orientation of the features given their relative positions. The measures thus satisfy the desired discriminative property as well.

3.3 Instance Recognition Algorithm

In instance recognition, the exact appearance of the object is known. This gives a tremendous amount of information to be leveraged into discriminative power. While "bag-of-words" approaches preserve some information from local features, information from the geometric configuration of said features is lost.

One example of geometric information that might be lost is that for *my* car, the ratio of the distance between the front and rear wheels to wheel size is four to one. For features purporting to correspond to the front and rear wheels, this ratio can be computed and compared to the known value.

Leveraging this type of geometric information by applying the "geometric grammar" developed in Section 3.2.4 enables filtering out both spurious features and real features from other instances. For example, a pair of features matching the front and rear wheels which have a ratio of three to one can be reliably rejected.

An effective filtering architecture can be constructed, taking the form of a cascade of filters. This cascade of filters would iteratively reject spurious features while

preserving the genuine ones. Each filter stage incorporates a voting mechanism. The final architecture along with parameter values are subsequently described. This architecture allows for a significant degree of robustness to image noise.

For the duration of discussion regarding instance recognition, *scene* refers to the test image over which we are searching for objects while *template* refers to the training image of the object which we are searching for. A detection is a match between a portion of the *scene* and a *template*.

3.3.1 Voting

Each filtering stage uses voting to establish whether a feature should be considered legitimate. The theory states that two features *close* to each other are likely to be part of the same object. The percentage of *close* neighbors which are *consistent* with a particular feature is a good indication to said feature being a legitimate match.

Below, we will explore two different voting mechanisms (NN and Distance-based) which apply different notions of *close* and *consistent*.

Why Not NN voting?

For nearest neighbor (NN) voting, *close* is defined to be the k nearest feature points. For a given match between the template and the scene, the set of k nearest neighbors to the feature in the scene and the set of k nearest neighbors to the feature in the template are considered. Each match between these two sets of k neighbors cast one vote for the match under consideration. Each vote is an additional piece of evidence of local *consistency*. If there are m or more matches, then the original match is considered legitimate. This voting mechanism is illustrated in Figure 3-4. Sivic and Zisserman's Video Google system used this method [39] as a light-weight heuristic to augment "bag-of-words". They used a value of $k = 15$ and $m = 1$.

This approach was demonstrated to work well under Video Google. However, it can still be improved on further, particularly for noisy videos. With noisy videos, there are often large clusters of spurious features generated. As a result, even genuine

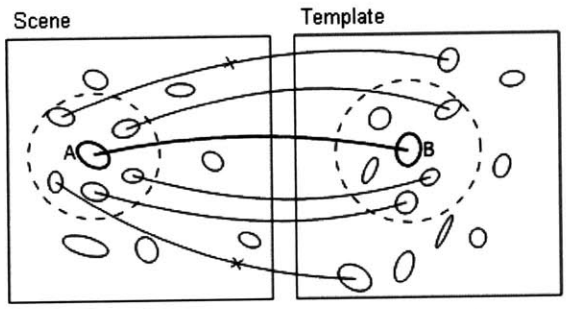

Figure 3-4: Illustration of NN spatial consistency voting. In this case, a match (A,B) is checked for consistency. The nearest $k = 5$ neighbors of each are inspected. Each match connects members of the two neighbor sets casts a positive vote. In this case, three other matches cast votes in support of (A,B). If there are fewer than m votes, (A,B) is rejected. This diagram is an excerpt from Sivic and Zisserman's Video Google [39].

features would not receive any support as their immediate neighborhood could be saturated with spurious matches. When this heuristic was applied data generated as described in 3.5.1, failure indeed occurred. This is largely a locality problem which can fixed by increasing the number of neighbors around the feature. Unfortunately, the cost of sorting makes it impractical to use for large values of k. A better method is to use a distance-based approach.

Distance-based voting

For distance-based voting, *close* is defined to be all feature points within a radius of R. This radius of R is the expected radius in pixels of the object of interest in the scene. For each match within R of the match being considered in the scene, a vote is cast in support if the matches are *consistent* according to the "geometric grammar." An illustration of this voting mechanism can be found in Figure 3-5. Note that in contrast to NN-voting, there is no explicit boundary on the template side. However, measures such as the ratio of the feature scale to distance would ensure matches such as 1 and 5 do not cast a vote. Measures such as relative orientation and relative scale would prevent matches such as 3 and 4 respectively from voting.

For genuine features on the object, all other genuine features within a radius R

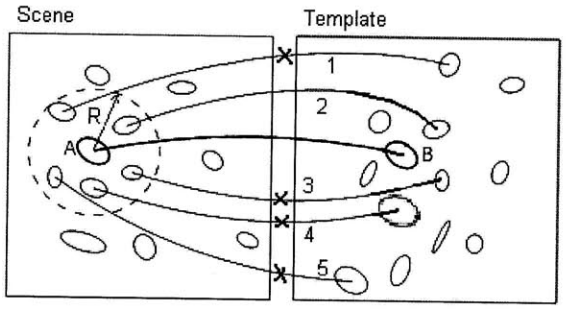

Figure 3-5: Illustration of distance-based spatial consistency voting. In this case, a match (A,B) is checked for consistency. Each match (C,D) within radius R of A in the scene is considered. If the pair of features A and C in the scene and the pair of features B and D in the template are considered to be consistent according to the "geometric grammar," then a vote is cast in support. Note in this case that only one matching vote is generated. The number of votes is normalized by number of potential votes to generate an approval rating (0.2 in this case). If the approval is lower than a threshold, then (A,B) is rejected

will match the geometric constraints while some spurious features within a radius of R will also match purely by chance (denote this likelihood by ϵ). Compare this against a spurious feature which would match other features within a radius of R solely on basis of chance ϵ.

Now consider the number of votes normalized by the total number of votes possible. This can be thought of as an approval rating. For genuine features, the approval rating would be a weighted average of 1 and ϵ. For spurious features, approval rating would be ϵ. It is reasonable to assume that ϵ will be small as each "geometric grammar" rule should be discriminative (the underlying geometric measure has a wide distribution, making the probability of any given value occurring by chance relatively unlikely). Thus, even with relatively few other genuine features in the radius, there should be a significant separation of approval ratings between genuine and spurious features. A threshold can then be imposed which take advantage of this separation to generate a binary decision.

Correct selection for the value of R is helpful but not essential. If R value is overestimated, then the fraction of genuine matches in the radius for a genuine feature would be reduced. This would result in reduced separation in approval ratings for

genuine and spurious matches. If R is underestimated, then there is relatively little impact. Granted if R is too small, the same locality problem which NN-voting suffers from occurs and sensitivity to noise increases.

3.3.2 Grammar Cascade

The face detection system of Viola and Jones [46] used a cascade of filters to filter out as many false positives as possible while preserving the true positives at each stage. This design has computational advantages over traditional boosting methods.

The cascade of "grammar rules" takes advantage of similar principles to save on computation. However, given the more discriminative nature of the "geometric grammar," fewer stages are necessary. The thresholds on approval ratings can be gradually turned up to reject as many false positives as possible while preserving the true positives.

3.3.3 Final Instance Recognition Algorithm

For each geometric measure g, define a distortion factor between the value in the template and the scene to be $\max(\frac{g(\text{template})}{g(\text{scene})}, \frac{g(\text{scene})}{g(\text{template})})$. Experimentally, for a pair of genuine feature matches, this distortion factor can be as large as 1.2. Most of the objects in the scenes had an approximate radius of 120 pixels or less. A threshold value of $R = 100$ for feature pairs to consider worked well.

The final cascade contains three stages of filtering by voting.

First Stage

The first stage generates positive votes for both features of any feature pair where both the distortion factors for the ratio of scales and the ratio of average scale to length of connecting line (first and second in the list in Section 3.2.4 respectively) are less than their respective thresholds $SCALE\_FUDGE\_FACTOR$ and $DIST\_SCALE\_FUDGE\_FACTOR$. For a feature to survive until the next stage, it must have an approval rating of at least $DIST\_APPROVE\_THRESHOLD$ (it's ge-

ometrically consistent with at least $DIST\_APPROVE\_THRESHOLD$ of the other features with distance R). The approval rating is the fraction of the number of votes received divided by the total number of potential votes. This normalization works better than just counting the number of votes received as it accounts for areas with dense features. With many features, even a spurious feature may be able to get votes by chance (ϵ). We also stipulate that the number of votes received must be at least two. This accounts for the other extreme where features are very sparse and a single positive vote may constitute a significant fraction of the total. If only one other feature voted, then the result can be unreliable and it is best to discard. This is illustrated in Algorithm 1.

Second Stage

The second stage generates positive votes for both features of any feature pair where both the distortion factors for signed difference in orientation and difference between average orientation and angle of connecting line segment (third and fourth in the list in Section 3.2.4 respectively) are less than their respective thresholds ANG_FUDGE_FACTOR and $ANG\_ORIENT\_FUDGE\_FACTOR$. The approval rating cutoff is $ANG\_APPROVE\_THRESHOLD$. This is illustrated in Algorithm 2.

Third Stage

The third stage is the same as the first stage with the difference that the approval rating cutoff is $RE\_DIST\_APPROVE\_THRESHOLD$. Note that this cutoff is larger than the cutoff used in the first stage $DIST\_APPROVE\_THRESHOLD$ and is hence more restrictive. While at first glance, it appears that we can save the computation in this stage by simply using the higher cutoff back in the first stage, this is not the case. Note that the space of inputs to the first stage and third stage is very different. In the first stage, most of the inputs are false positives with a few true positives mixed in. In the third stage, the ratio is reversed with the majority of inputs being true positives. Due to the definition of approval rating as number

Algorithm 1: Final Instance Recognition Algorithm First Filtering Stage

Data: L_s (descriptor locations in the scene), L_m (descriptor locations in the model), M is a mapping $L_s \rightarrow L_m$ where M is a partial function of L_s. In practice, for any $l_s \in L_s$ where there is no corresponding $l_m \in L_m$, $M(l_s) = 0$.

Result: $M_{dist\_filtered}$ (correspondences which are consistent with the "geometric grammar" at the first stage)

$s \leftarrow$ number of descriptors in L_s;
initialize *votes* to be 0 filled vector of length s;
initialize *possible_votes* to be 0 filled vector of length s;
for $i \leftarrow 1$ **to** s **do**
  for $j \leftarrow 2$ **to** s **do**
    if $L2(L_s(i),L_s(j)) < R$ **then**
      $possible\_votes(i) \leftarrow possible\_votes(i) + 1$;
      $possible\_votes(j) \leftarrow possible\_votes(j) + 1$;
      $sceneScale \leftarrow$ `Scale` $(L_s(i),L_s(j))$;
      $modelScale \leftarrow$ `Scale` $(L_m(M(i)),L_m(M(j)))$;
      $scaleFudgeFactor \leftarrow \max(\frac{sceneScale}{modelScale}, \frac{modelScale}{sceneScale})$;
      if $scaleFudgeFactor < SCALE\_FUDGE\_FACTOR$ **then**
        $sceneScaleDist \leftarrow$ `ScaleDist` $(L_s(i),L_s(j))$;
        $modelScaleDist \leftarrow$ `ScaleDist` $(L_m(M(i)),L_m(M(j)))$;
        $scaleDistFudgeFactor \leftarrow \max(\frac{sceneScaleDist}{modelScaleDist}, \frac{modelScaleDist}{sceneScaleDist})$;
        if $scaleDistFudgeFactor < DIST\_SCALE\_FUDGE\_FACTOR$ **then**
          $votes(i) \leftarrow votes(i) + 1$;
          $votes(j) \leftarrow votes(j) + 1$;
        end
      end
    end
  end
end
for $i \leftarrow 1$ **to** s **do**
  if $votes(i) >= 2 \&\& votes(i)/possible\_votes(i) > DIST\_APPROVE\_THRESHOLD$ **then**
    $M_{dist\_filtered}(i) \leftarrow M(i)$;
  else
    $M_{dist\_filtered}(i) \leftarrow 0$;
  end
end

Algorithm 2: Final Instance Recognition Algorithm Second Filtering Stage

Data: L_s (descriptor locations in the scene), L_m (descriptor locations in the model), $M_{dist\_filtered}$ is a mapping $L_s \rightarrow L_m$ where $M_{dist\_filtered}$ is a partial function of L_s. In practice, for any $l_s \in L_s$ where there is no corresponding $l_m \in L_m$, $M_{dist\_filtered}(l_s) = 0$.

Result: $M_{ang\_filtered}$ (correspondences which are consistent with the "geometric grammar" at the second stage)

$s \leftarrow$ number of descriptors in L_s;
initialize *votes* to be 0 filled vector of length s;
initialize *possible_votes* to be 0 filled vector of length s;
for $i \leftarrow 1$ **to** s **do**
  for $j \leftarrow 2$ **to** s **do**
    if $L2(L_s(i),L_s(j)) < R$ **then**
      $possible\_votes(i) \leftarrow possible\_votes(i) + 1$;
      $possible\_votes(j) \leftarrow possible\_votes(j) + 1$;
      $sceneAngle \leftarrow$ `Angle` $(L_s(i),L_s(j))$;
      $modelAngle \leftarrow$ `Angle` $(L_m(M_{dist\_filtered}(i)),L_m(M_{dist\_filtered}(j)))$;
      $angleFudgeFactor \leftarrow \max(\frac{sceneAngle}{modelAngle}, \frac{modelAngle}{sceneAngle})$;
      if $angleFudgeFactor < ANG\_FUDGE\_FACTOR$ **then**
        $sceneAngleOrient \leftarrow$ `AngleOrientation` $(L_s(i),L_s(j))$;
        $modelAngleOrient \leftarrow$ `AngleOrientation`
        $(L_m(M_{dist\_filtered}(i)),L_m(M_{dist\_filtered}(j)))$;
        $angleOrientFudgeFactor \leftarrow$
        $\max(\frac{sceneAngleOrient}{modelAngleOrient}, \frac{modelAngleOrient}{sceneAngleOrient})$;
        if $angleOrientFudgeFactor <$
        $ANG\_ORIENT\_FUDGE\_FACTOR$ **then**
          $votes(i) \leftarrow votes(i) + 1$;
          $votes(j) \leftarrow votes(j) + 1$;
        end
      end
    end
  end
end
for $i \leftarrow 1$ **to** s **do**
  if $votes(i) >= 2 \&\& votes(i)/possible_votes(i) >$
  $ANG\_APPROVE\_THRESHOLD$ **then**
    $M_{ang\_filtered}(i) = M_{dist\_filtered}(i)$;
  else
    $M_{ang\_filtered}(i) = 0$;
  end
end

of votes divided by number of possible votes, even the true positives cannot score very high when there are many false positives dragging down their score. Hence, using a higher cutoff in the first stage would eliminate significant number of true positives as well as false positives. In the third stage, however, the approval ratings for true positives are much higher and we can utilize this to further reduce the few false positives that remain. This is illustrated in Algorithm 3.

MLESAC

After the filter cascade, a RANSAC variant called MLESAC developed by Torr and Zisserman [41] was used to recover the rigid body Rotation Scale Translation (RST) transform which would map the locations of the features in the matching template into the test scene. The interest points from the template can be projected via the recovered transform to the scene. This produces a good estimate of the location of the object in the scene. For ease of visualization, a fairly accurate bounding box can be drawn around these points. The MLESAC procedure also throws out some outliers which are not geometrically consistent with the other feature points.

Non-maximal Suppression

The above steps are performed for many different templates of any given object at varying orientations formed by out-of-plane rotations. If we stopped here, then we would be faced with the issue of having many virtually identical bounding boxes for each object. This is due to the fact that SIFT and our geometric constraints possess some out-of-plane invariance. As a result, many neighboring orientations also tend to be triggered. A form of non-maximal suppression can be used to eliminate the duplication. First, a list of templates for various orientations is sorted in descending order by the number of valid feature matches. For each template, the rigid body RST transform from MLESAC is used to compute the 2D location of the center of the detection projected into the scene. A check is added to test whether or not this center is closer than a threshold to a detection that is already added for this frame and object identity. If so, then this result is discarded.

Algorithm 3: Final Instance Recognition Algorithm Third Filtering Stage

Data: L_s (descriptor locations in the scene), L_m (descriptor locations in the model), $M_{ang\_filtered}$ is a mapping $L_s \rightarrow L_m$ where $M_{ang\_filtered}$ is a partial function of L_s. In practice, for any $l_s \in L_s$ where there is no corresponding $l_m \in L_m$, $M_{ang\_filtered}(l_s) = 0$.

Result: $M_{re\_dist\_filtered}$ (correspondences which are consistent with the "geometric grammar" at the first stage)

$s \leftarrow$ number of descriptors in L_s;
initialize *votes* to be 0 filled vector of length s;
initialize *possible_votes* to be 0 filled vector of length s;
for $i \leftarrow 1$ **to** s **do**
  for $j \leftarrow 2$ **to** s **do**
    if $L2(L_s(i),L_s(j)) < R$ **then**
      $possible\_votes(i) \leftarrow possible\_votes(i) + 1$;
      $possible\_votes(j) \leftarrow possible\_votes(j) + 1$;
      $sceneScale \leftarrow$ `Scale` $(L_s(i),L_s(j))$;
      $modelScale \leftarrow$ `Scale` $(L_m(M_{ang\_filtered}(i)),L_m(M_{ang\_filtered}(j)))$;
      $scaleFudgeFactor \leftarrow \max(\frac{sceneScale}{modelScale}, \frac{modelScale}{sceneScale})$;
      if $scaleFudgeFactor < SCALE\_FUDGE\_FACTOR$ **then**
        $sceneScaleDist \leftarrow$ `ScaleDist` $(L_s(i),L_s(j))$;
        $modelScaleDist \leftarrow$ `ScaleDist` $(L_m(M_{ang\_filtered}(i)),L_m(M_{ang\_filtered}(j)))$;
        $scaleDistFudgeFactor \leftarrow \max(\frac{sceneScaleDist}{modelScaleDist}, \frac{modelScaleDist}{sceneScaleDist})$;
        if $scaleDistFudgeFactor < DIST\_SCALE\_FUDGE\_FACTOR$ **then**
          $votes(i) \leftarrow votes(i) + 1$;
          $votes(j) \leftarrow votes(j) + 1$;
        end
      end
    end
  end
end
for $i \leftarrow 1$ **to** s **do**
  if $votes(i) >= 2 \&\& votes(i)/possible\_votes(i) > RE\_DIST\_APPROVE\_THRESHOLD$ **then**
    $M_{re\_dist\_filtered}(i) \leftarrow M_{ang\_filtered}(i)$;
  else
    $M_{re\_dist\_filtered}(i) \leftarrow 0$;
  end
end

3.3.4 An Optional Improvement to Filtering: Uniqueness of Supporting Model Bigrams

The filtering cascade relies upon the concept that valid features will form bigrams which are consistent with the "geometric grammar". The measure of a feature's validity is based upon the percentage of neighboring features with which the feature in question forms grammatical bigrams. Each such bigram generates a vote of support. Furthermore, there is the stipulation that there must be more than one supporting vote. Those with only one supporting vote are considered unreliable. The bigrams being considered here are all located in the test scene.

During experimentation, one vexing scenario occurred which is illustrated in Figure 3-6. Particularly in image containing regular, repeated patterns such as windows on a building, it is possible for many features in the scene to match the same feature in the template. In this case, we have a cluster of 2 and a cluster of 4. A single mis-association error can thus be amplified into a cluster of false positive matches. What is particularly problematic is when two (or more) such clusters are located near each other. Each false positive match in a cluster will generate votes of support for matches in the other clusters. Depending on the size of the clusters, the number of supporting votes thus generated can match or even exceed votes for legitimate features. In the case of Figure 3-6, all the true positives get 3 votes while a false positive cluster of 4 will generate 4 votes for each match in the other cluster. This condition makes a global threshold on the number of supporting votes unreliable.

Upon further inspection, we note that while the false positives form large clusters in the scene, they only correspond to single feature in the template. This suggests a revision to only count the number of votes which come from features that map to distinct features in the template. This results in the false positives receiving support from only one distinct feature while the true positives would receive support from multiple distinct features (3 in this case).

The improvement in robustness gained from enforcing this additional constraint comes at the cost of roughly 10% increase in the run-time of the filter. Thus, when

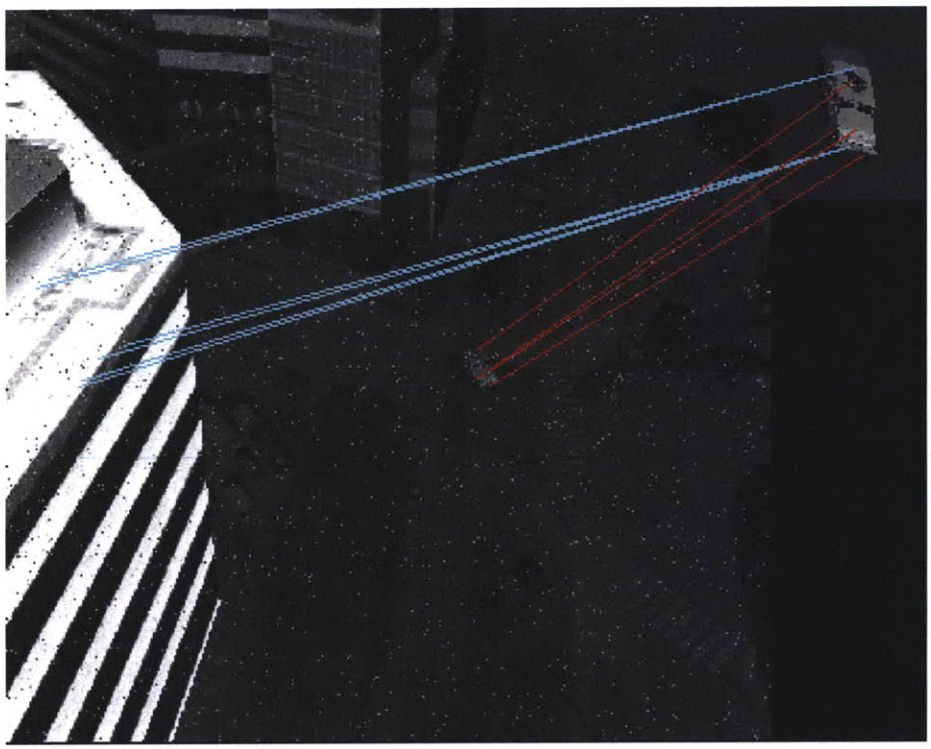

Figure 3-6: If only the bigrams in the scene are considered, the false positives (cyan) can match or exceed the true positives (red) in the number of supporting matches. One false positive gets 4 votes of support while all the true positives get 3. Appropriate fix is to only count support from ones which map to distinct features in the template. When that is done, the false positives receive only 1 votes while the true positives receive 3. A threshold can then be applied.

this additional improvement is used, it is recommended to apply it only to the last stage of the cascade. Doing so produces most of the same benefits with minimal impact to run time. This can be observed from the fact that each layer of the feature cascade aggressively culls out false positives which reduces input size and consequently run-time dramatically as we move into the later stages of the cascade.

3.4 Category Recognition Algorithm

In contrast to instance recognition, there is no longer a canonical model/template for the object of interest in the case of category recognition. However, some geometric information in the form of "geometric measures" can still be useful as the measures may come from different underlying distributions for different classes. Below, we discuss an example of how "geometric measures" could be useful in category recognition. We then propose a method for modeling distributions of geometric measures and integrate this with a pLSA "bag-of-words" model to produce a "bag-of-phrases" model.

3.4.1 Example of Utility of Geometric Measures in Category Recognition

Suppose that instead of identifying a specific car as in the instance recognition case (Section 3.3), the task is to identify whether a vehicle is a car or a tractor. Consider a pair of features corresponding to the front and rear wheels respectively. For a car, the relative scale ratio might follow some distribution which is centered around 1 as most cars tend to have similar sized wheels. For a tractor, the ratio might have a mode around 2 or 3 since many tractors have larger rear wheels than front wheels. Information about the relative size of the parts (wheels) can serve as a good indicator for the category (car or tractor) an object (vehicle) belongs to.

3.4.2 Modeling the Distribution of Geometric Measures

In general, a given geometric measure such as the ratio of the wheel sizes having a value of 2 can be rewritten as $g(w_1, w_2) = a$ to emphasize the fact that the geometric measure in question is a function of two "visual words" taking on a particular value. In order to model the likelihood of seeing a certain object category based upon whether the observed bigrams follow some "geometric grammar", we first need to learn the "geometric grammar" for that category. This grammar is separated into different rules for each "geometric measure" where the expected values can be modeled by the probability $P((g(w_1, w_2) = a)|z)$ where z represents the latent variable denoting object category.

$g(w_1, w_2)|z$ is an unknown continuous distribution. From training data, it's possible to obtain samples $a_{z1}, a_{z2}, ...a_{zn}$ for the value of $g(w_1, w_2)$ from this distribution for a given z. The distribution $P((g(w_1, w_2) = a)|z)$ which is recovered should have a property such that if there are many a_{zi} close to a, then the probability returned will be high. One way in which this can be done is through a "gravitational model" where the existing sample a_{zi} would pull a in one direction or another, acting with significantly more effect if they are close. Without a prior, all samples should have equal weight. This proposed heuristic for approximating $P((g(w_1, w_2) = a)|z)$ is shown in Equation 3.1. Note that the amount of attraction is capped at 10000 to prevent any values from becoming unbound.

$$P((g(w_1, w_2) = a)|z) \propto \frac{1}{n} \sum_i \min(\frac{1}{(a - a_{zi})^2}, 10000) \tag{3.1}$$

One question which may come to mind is why not use a Gaussian to model $g(w_1, w_2)|z$? Equation 3.1 would require storing all the samples as well as take lots of computation. Meanwhile, a Gaussian would only require maintaining the values for the sample mean and sample standard deviation.

While these are valid points, assuming the underlying distribution is Gaussian (an unimodal distribution) would likely result in a gross simplification. In particular, note that the distributions need not be unimodal. Going back to the tire size ratio

example, there may be several valid tractor tire ratios while anything in between cannot be a tractor.

One particular optimization can prove fruitful without oversimplification. This entails merging measurements together heuristically and weighting them to reduce the computational costs when the number of samples grows large. This can be rationalized as merging nearby measurements/bodies together to produce ones of greater mass.

3.4.3 Bag-of-Phrases

While a traditional "bag-of-words" pLSA model considers the conditional probability of single "words" $P(w|z)$, the proposed "bag-of-phrases" model considers the joint conditional probability of pairs of "words" $P(w_1, w_2|z)$. Note that this joint conditional probability must also take into account the probabilities associated with the geometric measures relating to this pair (w_1, w_2). For this, we assume that the features appear independently and also the value of the geometric features are independent. The first assumption is convenient computationally while the second is justified from an information theoretical point of view as all the geometric measures should be necessary for fully determining the relative geometric configuration of the features as noted in section 3.2.3. The joint conditional probability is shown in Equation 3.2.

$$P(w_1, w_2|z) = P(w_1|z)P(w_2|z)\prod_g P(g_j(w_1, w_2) = a_i|z) \tag{3.2}$$

In Equation 3.2, g is the set of various geometric features and a_i are the specific values that they may take on.

The overall probability for the category of interest appearing in the image $P(d|z)$ can thus be expressed as in Equation 3.3.

$$P(d|z) = \sum_{w_1,w_2 \in W(z), dist(w_1,w_2) \leq R} P(w_1, w_2|z) \tag{3.3}$$

In Equation 3.3, d is the category label, $dist$ is a distance function (L2 in this case), R is an approximate radius for what is considered semi-local and $W(z)$ are the words

that have the highest $P(w|z)$ in the vocabulary. The set $W(z)$ is learned using pLSA. The underlying intuition states that the most frequently occurring "visual words" for any category are most likely to represent semantically meaningful parts of an object. Consequently "geometric grammar" rules which link them are most likely to result in larger semantically meaningful parts/sub-assemblies of the object or define some intrinsic property of the object.

3.5 Instance Recognition Evaluation

For instance recognition, there are extremely powerful constraints due to the knowledge of geometric layout of features. In a series of experiments, we will explore the limits of how an end-to-end system might be able to perform on recognition and localization of known objects under adverse conditions such as large amounts of illumination change, clutter, and noise.

3.5.1 Experimental Setup

Ground truth was needed for demonstration of eventual end-to-end capability. In addition, a dataset which closely resembled actual visual operating conditions was desired. Due to lack of availability of a suitable dataset, a custom one was generated using computer graphics.

A high resolution CAD model of the Technology Square area was obtained. This served as the backdrop for test scenario to represent an urban scene. Four CAD models of vehicles (Humvee, Mig, Abrams, and Apache) were downloaded from the web. These vehicles served as the objects of interest to be searched for.

An application was developed based upon the OpenSceneGraph library (an Open Source object rendering/simulation framework built on top of OpenGL) to read in CAD models and render images containing them from arbitrary relative positions and orientations. This application was then used to generate both templates and test data.

The templates were generated by taking snapshots of the CAD vehicles at 10^o

intervals of out-of-plane rotation against an uniformly colored background. The test data was generated by placing the CAD models into the Tech Square model and then flying the camera around, rendering from different position and orientations. Salt-and-pepper noise, which corrupted up to 15% of the pixels, was also added to some of the testing data in order to simulate the type of imaging noise that may be found in a real camera.

Due to the fact that this data was generated, information such as the position and orientation of the camera at each frame is known (this information is leveraged in later chapters discussing tracking and localization). Furthermore, on the template side, the relative homography transforms between the different orientations is also known.

3.5.2 Research Questions

Listed below are a number of research questions which we would like to answer through experiments in the following sections.

1. How well does raw SIFT feature matching ("bag-of-words" style) work in localizing an object in the scene given the correct orientation a priori? This can be considered a baseline.

2. How well does the proposed filter cascade approach work on the same problem and how does this compare with the baseline?

3. Is it possible to obtain the same level of performance by varying the parameters with raw SIFT feature matching?

4. How fast does all of this run?

5. Can the filter cascade differentiate between different orientations of the same object?

6. Can we detect several different object instances under a variety of adverse imaging conditions?

7. Can we detect real objects in real scenes by matching against CAD model templates?

3.5.3 Baseline and Example

Consider the scene in Figure 3-7. This will serve as a running example through the next few sections.

Figure 3-7: Running example of a scene over which we need to search for objects. Note that this image is noisy and contains not only a significant amount of illumination variation but also some compression artifacts.

We want to find the Humvee in the scene as well as the orientation it is at. In this case, the desired orientation is displayed in Figure 3-8. The desired orientation has an off-nadir angle of 60^o (denote this as latitude) and an azimuth angle of 150^o (denote this as longitude). In this case, nadir is top-down and azimuth is from back to front of the vehicle.

For a baseline, we will use David Lowe's method [31] of assigning a novel test feature to a known feature if the distance between the two is the smallest compared

Figure 3-8: Desired orientation of the Humvee that we are looking for in the scene shown in Figure 3-7. This assumes invariance to in-plane rotation.

to all other known features and if the ratio of this distance to the second closest distance is below a certain threshold. Let us denote this threshold as the SIFT matching threshold. A typically recommended SIFT matching threshold is 0.8. Note that in this case, we allow each feature in the scene to have the opportunity of matching a feature in the template. However, it is perfectly fine for a feature in the template to match multiple features in the scene as potentially more than one instance of the object can be present in the scene. The desired result has as many true positives as possible with as few false positives as possible. Such a result with very high signal-to-noise ratio would be the most useful to a histogram based approach like "bag-of-words".

In Figure 3-9, we can see the result for baseline matching with no filtering. There are 80 total matches of which 10 are true positives. This produces a signal-to-noise ratio of only $\frac{1}{8}$. Any attempts to use this histogram would likely result in matches to many things which are not the object of interest. For reference, there are a total of 1270 features in the scene and 49 features in the template. SIFT does retrieve true positives at better than chance. Now, we would like to see if we can improve the signal-to-noise ratio further.

3.5.4 Results with Filtering Cascade

The filter cascade can be applied to produce the desired improvement in signal-to-noise ration by eliminating false positives while maintaining true positives.

First, we initialize the cascade with some parameter values shown in Table 3.1.

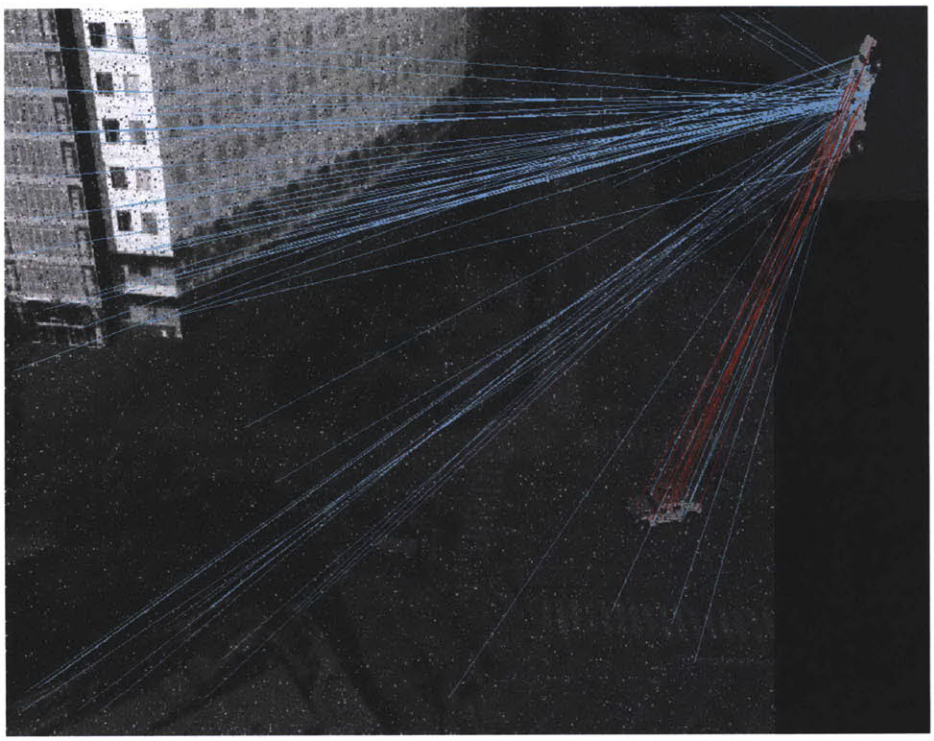

Figure 3-9: Baseline (no filtering) for SIFT matching threshold at 0.8 shows poor signal-to-noise ratio. There are many false positives (cyan) and comparatively few true positives (red).

As we will see later, exact parameter settings are not important.

Parameter Name	Value
R	75
$SCALE\_FUDGE\_FACTOR$	1.45
$DIST\_SCALE\_FUDGE\_FACTOR$	1.45
$DIST\_APPROVE\_THRESHOLD$	0.2
ANG_FUDGE_FACTOR	1.5
$ANG\_ORIENT\_FUDGE\_FACTOR$	0.6
$ANG\_APPROVE\_THRESHOLD$	0.4
$RE\_DIST\_APPROVE\_THRESHOLD$	0.6

Table 3.1: Some parameter values

In Figure 3-10, we can see that successive layers of filtering will eliminate all false positives while preserving all the true positives. This greatly increases the signal-to-noise ratio. The number of true positives and false positives is summarized in Table 3.2.

Filtering	Matches	True Positives	False Positives	TPR	FPR
None	80	10	70	10/49	70/1221
First	15	10	5	10/49	5/1221
Second	10	10	0	10/49	0/1221
Third	10	10	0	10/49	0/1221

Table 3.2: Filtering Cascade Performance Summary (0.8 SIFT Matching Threshold)

3.5.5 Varying SIFT Match Threshold

The previous section demonstrated a significant improvement to the signal-to-noise ratio by using the filter cascade to eliminate all the false positives. The question remains as to whether the same level of false positive rejection can be achieved by just varying the SIFT matching threshold. Furthermore, while loosening up the SIFT matching threshold can produce additional true matches to aid in detection, this comes at the cost of many false matches. We would like to see if the filter cascade can help us eliminate these false matches. To answer these questions, we experiment with several different values for this threshold: 0.6, 0.7, 0.8, 0.85, 0.9, and 0.95.

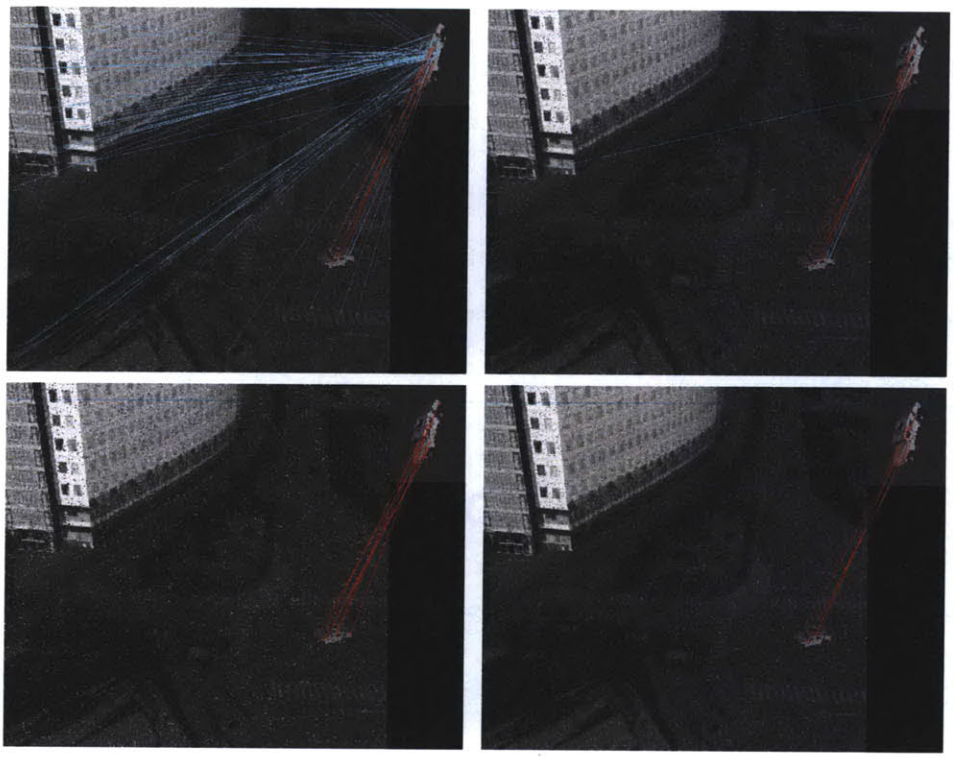

Figure 3-10: Successive layers of filtering (clockwise from top-left) remove all false positives (cyan) while preserving all the true positives (red). Red lines denote true positives, cyan false positives.

As we can see in Figure 3-11, using the filter cascade, we are able to filter out every false positive without losing a single true positive. More detailed statistics can be found in Table 3.3. The number of apparent features in the images may differ from the table. This is due to the fact that multiple features may co-occur at a single location. In all cases, the number in the table reflects the true number of feature matches.

Thresh	Filtering	Matches	True Positives	False Positives	TPR	FPR
0.6	None	4	4	0	4/49	0/1221
0.6	First	4	4	0	4/49	0/1221
0.6	Second	4	4	0	4/49	0/1221
0.6	Third	4	4	0	4/49	0/1221
0.7	None	17	7	10	7/49	10/1221
0.7	First	10	7	3	7/49	3/1221
0.7	Second	7	7	0	7/49	0/1221
0.7	Third	7	7	0	7/49	0/1221
0.8	None	80	10	70	10/49	70/1221
0.8	First	15	10	5	10/49	5/1221
0.8	Second	10	10	0	10/49	0/1221
0.8	Third	10	10	0	10/49	0/1221
0.85	None	166	10	156	10/49	156/1221
0.85	First	20	10	10	10/49	10/1221
0.85	Second	10	10	0	10/49	0/1221
0.85	Third	10	10	0	10/49	0/1221
0.9	None	304	11	293	11/49	293/1221
0.9	First	28	11	17	11/49	17/1221
0.9	Second	11	11	0	11/49	0/1221
0.9	Third	11	11	0	11/49	0/1221
0.95	None	620	13	607	13/49	607/1221
0.95	First	40	17	23	13/49	23/1221
0.95	Second	14	13	1	13/49	1/1221
0.95	Third	13	13	0	13/49	0/1221

Table 3.3: Filter cascade eliminates all false positives without loss of true positives at all thresholds

From the table, we can see that the filter cascade is capable of filtering out all the false positives without eliminating any true positives. Is it possible to accomplish this without using any filter? In this case, setting the SIFT matching threshold to

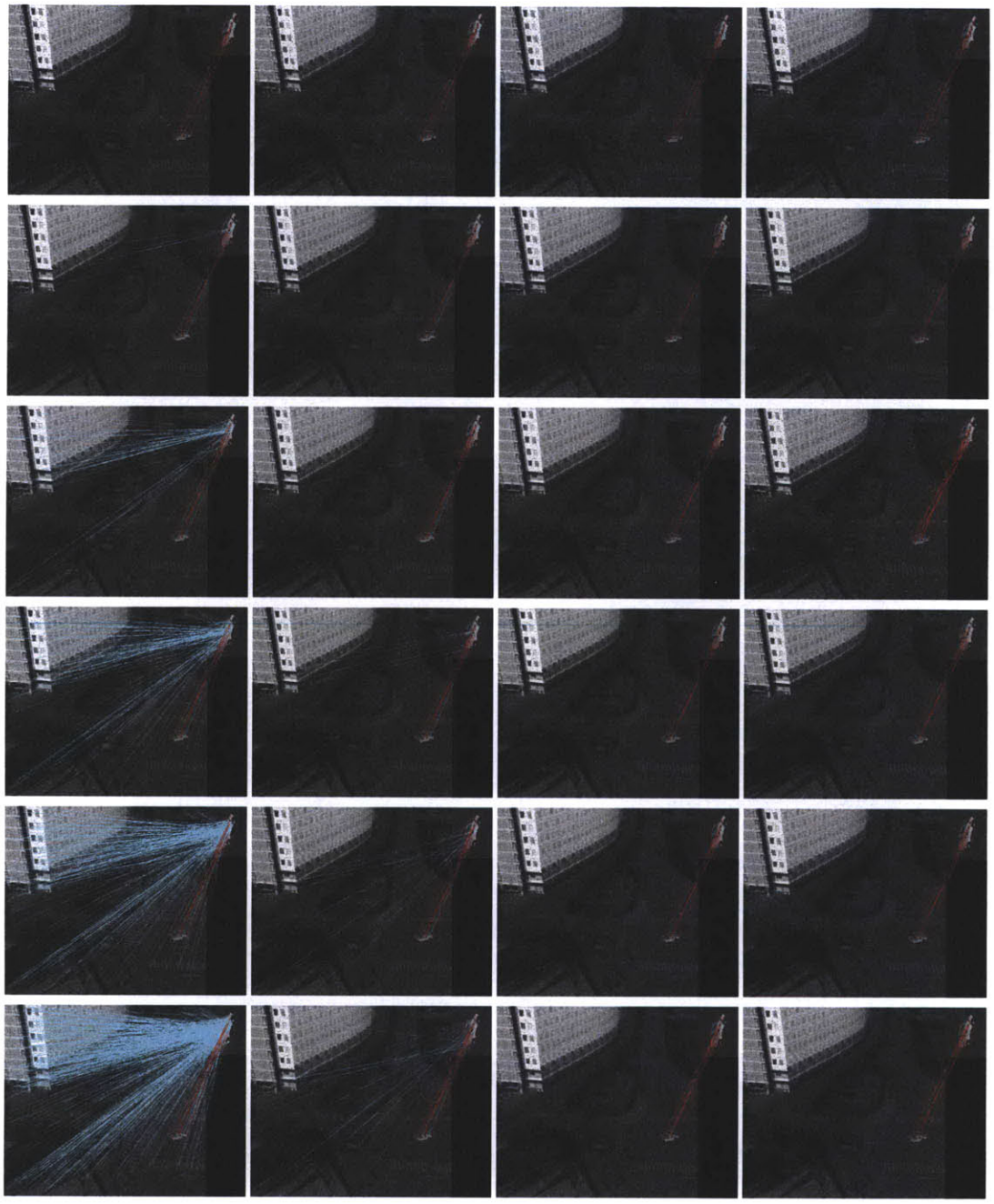

Figure 3-11: For all SIFT matching thresholds (from top: 0.6, 0.7, 0.8, 0.85, 0.9, 0.95), the filtering cascade (increasing filtering across) is able to eliminate all the false positives (cyan) without losing any true positives (red).

0.6 would do so. However, this would result in only 4 true positive correspondences. Loosening this threshold, we can get up to 13 true positive correspondences. These additional correspondences would give us even more robustness against imaging conditions, occlusions etc. Thus, there are benefits to applying the filter cascade.

We can see from Table 3.3 that most of the false positives are eliminated by the first stage filter. The second stage filter usually removes the rest. Only in the 0.95 case did the third stage filter have to remove any false positives. In all cases, the three stages of filtering removed all the false positives.

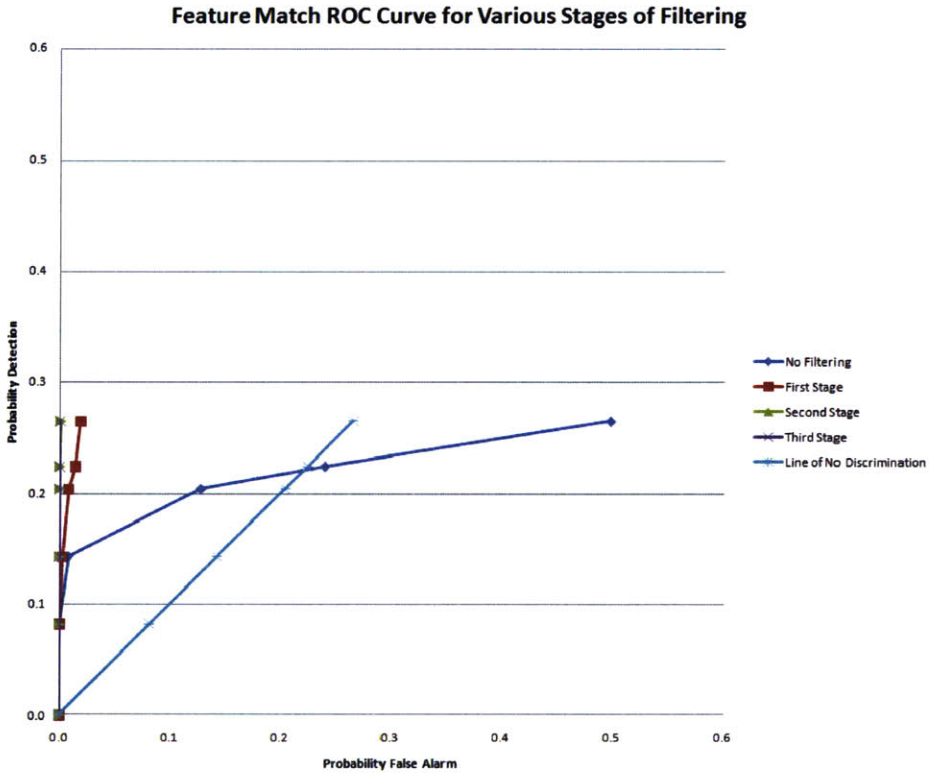

Figure 3-12: ROC curves for different stages of filtering. For the same detection level, application of the filters result in significantly lower false alarm rate. For SIFT matching threshold of 0.95, the filter cascade can even bring the value from the other side of the line of no discrimination over to the vertical axis. Due to significant image noise, even with very loose matching thresholds, no more than roughly 27% of the Humvee features can be recovered

Each successive stage of the filter cascade has the effect of moving to a different ROC curve with lower false alarm probability for any given detection level. For an illustration of this, see Figure 3-12. For SIFT matching threshold of 0.95, the filter

cascade can even bring the value across the line-of-no-discrimination to the vertical axis. This demonstrates the power of using these constraints.

In our running Humvee example, a SIFT matching threshold of 0.6 is sufficient for detection. This detection can be visualized in the form of a bounding box. At a threshold of 0.6, a reasonable bounding box is attainable without filtering. However, in other cases, this is just not possible. The result of searching for a MiG in another image is shown in Figure 3-13. In this case, we can see that all three stages of the filter cascade and also MLESAC were necessary to eliminate all the false positives. SIFT matching thresholds of 0.6 and 0.7 were not shown as they failed to produce enough matches to produce a non-degenerate bounding box that indicates a detection. For thresholds of 0.8-0.95, we were able to eliminate all the false positives which numbered at least 47. We were at the same time able to preserve all the true positives except at 0.95 where a single true positive was eliminated. It is worth noting that even with a single true positive eliminated, a threshold of 0.95 still produces a higher number of true positives than any other threshold without introducing any false positives. More detailed statistics are presented in Table 3.4.

In summary, the two examples in this section shows that increased SIFT matching threshold result in more true matches thus increasing robustness. High thresholds, however, would produce many false matches at the same time. "Bag-of-words" approaches would be unable to deal with this. In contrast, the filter cascade presented is able to filter out all the false matches while preserving the vast majority of true matches. This allows for greatly increased robustness in the presence of image noise.

	Before Cascade			After Whole Cascade		
SIFT Thresh	Matches	TP	FP	Matches	TP	FP
0.8	53	6	47	6	6	0
0.85	130	6	124	6	6	0
0.9	270	6	264	6	6	0
0.95	566	8	558	7	7	0

Table 3.4: SIFT Filtering on MiG Performance Summary (TP and FP denote true positives and false positives)

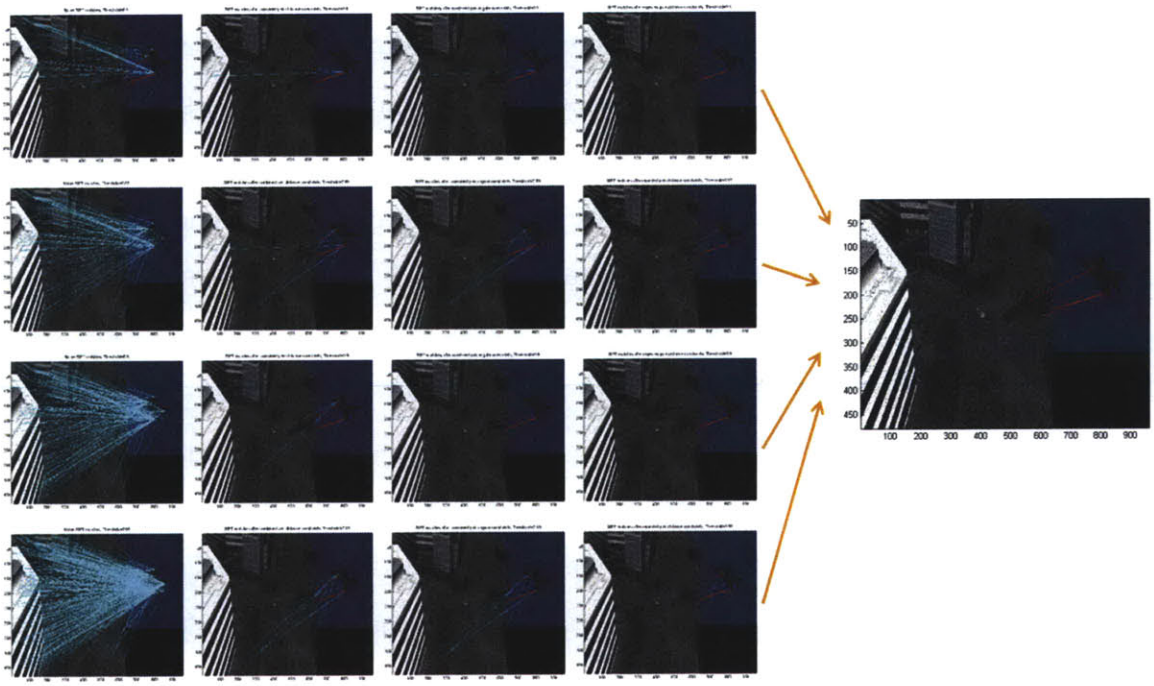

Figure 3-13: Across all SIFT matching thresholds (top down: 0.8, 0.85, 0.9, and 0.95), virtually identical results were produced after successive stages of filter cascade and MLESAC (across). All false positives (cyan) were rejected at the cost of at most one true positive (red)

3.5.6 Speed

The previous sections have demonstrated the ability of the filter cascade to generate a significant improvement in signal-to-noise ratio across all SIFT matching thresholds. This greatly improves robustness. The question remains as to whether this new found robustness comes at a significant hit to the running speed of the system.

The impact on run time for our running Humvee example can be found in Table 3.5 (results are averaged from 3 runs). The running time is largely dominated by MLESAC. At 0.9, MLESAC had some numerical instabilities. Such numerical instabilities may occur with a particular set of image correspondences. A different method of computing the transform may be able to eliminate such issues. Other than that, loosening the threshold drives up the cost of computation as expected. Therefore, although the filter cascade can eliminate hundreds of false positives, it is best not to rely on that if possible for the sake of speed. Solving for the transform with MLESAC can eliminate some false positives. However, it would be unwise to rely on that with no previous filtering. With the extreme numbers of false positives we are dealing with, the maximal consensus set may not even be that of the true positives. Furthermore, the runtime for MLESAC scales poorly with additional inputs. This would be unacceptable as it is already a bottleneck in the current implementation.

One note about current state of implementations: SIFT matching is done on a GPU, cascade filtering is performed in C++, and MLESAC is computed in Matlab. Porting the filters and MLESAC to the GPU should speed things up further and change the relative run times. The fact that the filter cascade is a principled and more efficient way than MLESAC to remove the false positives is nevertheless independent of implementation.

3.5.7 Differentiating Between Different Orientations

The previous sections have demonstrated the ability for the filter cascade to greatly improve the localization of an object of interest when the orientation is known a priori. In this section, we will consider the dual problem. Specifically, we would like

SIFT Thresh.	Matching	Pair Dist	Pair Ang	Pair Dist2	MLESAC	Total
0.6	0.00055	0.000088	0.000077	0.000051	0.0092	0.0100
0.7	0.00056	0.000094	0.000087	0.000065	0.0093	0.0101
0.8	0.00058	0.00020	0.000095	0.000065	0.0091	0.0100
0.85	0.00057	0.00085	0.000094	0.000073	0.0094	0.0110
0.9	0.00062	0.0037	0.0001	0.0001	0.3453	0.3498
0.95	0.00058	0.0202	0.0002	0.0001	0.0093	0.0304

Table 3.5: SIFT Humvee Filtering Speed Breakdown (in seconds)

to know the degree of accuracy to which can we determine the correct orientation of an object. To do so, we can scan through all the different orientations of the Humvee with the scene and see which ones end up with the most matches. We will use a SIFT matching threshold of 0.8. The results can be visualized in Figure 3-14.

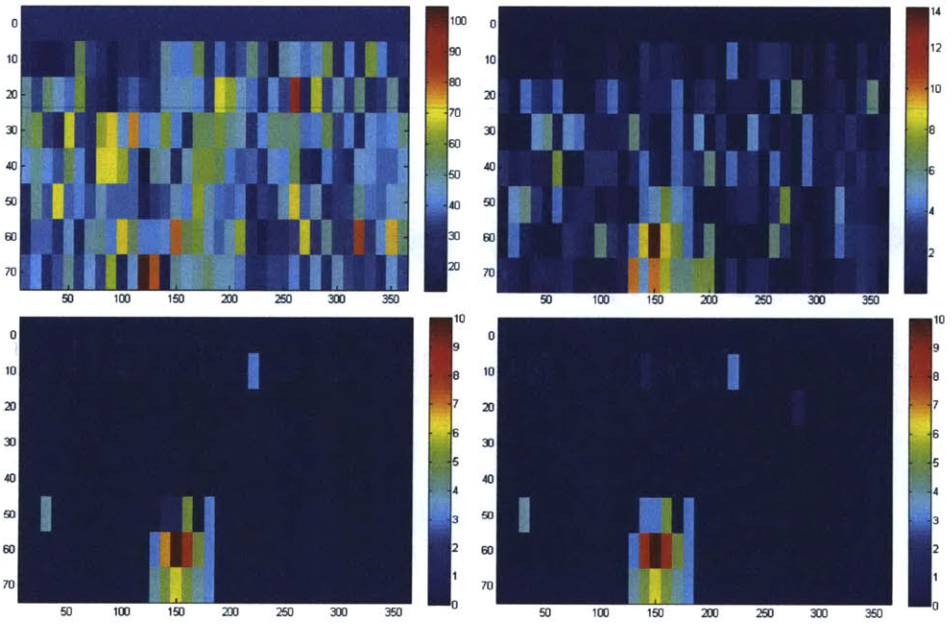

Figure 3-14: Heatmap representation shows a coherent cluster of orientations with many matches forming after successive stages of filtering (clockwise from top left).

Figure 3-14 shows the highest ranking orientations become more and more a coherent cluster with more filtering. This matches the expectation that the orientations around the correct one should score the highest. Note that the ground truth orientation as noted in Section 3.5.3 is consistently the highest after the first stage of

filtering. However, said orientation is not ranked the highest with no filtering. Thus, the bag-of-words approach would have mislead us regarding the correct orientation.

Figure 3-15 shows the same data in 3D form. With more filtering, the spikes inflated by false positive matches are literally cut down to produce a single, coherent peak near the correct result.

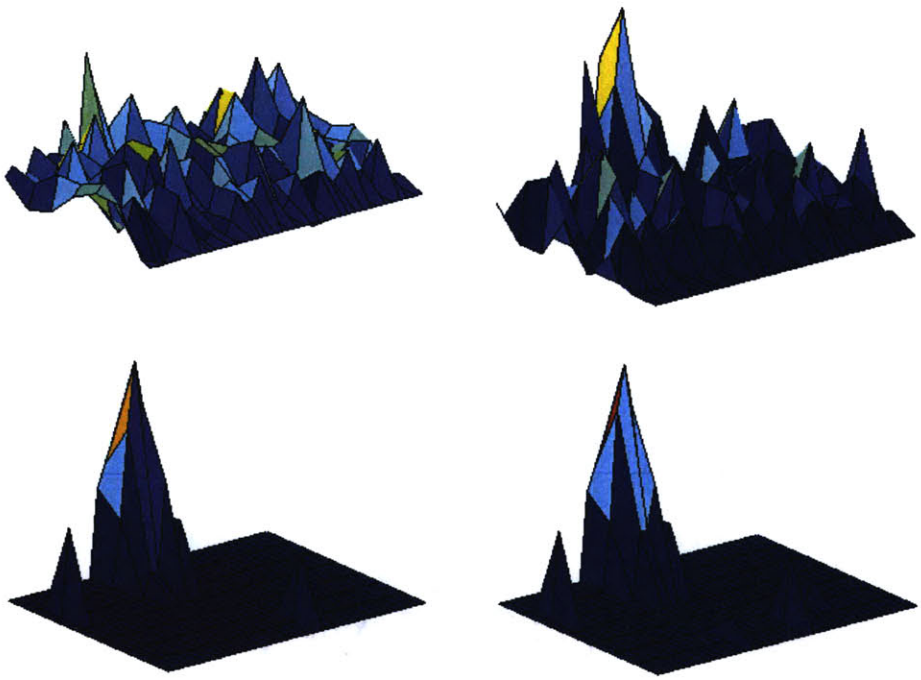

Figure 3-15: 3D plot shows a coherent cluster of orientations with many matches (showing as a large spike) forming after successive stages of filtering (clockwise from top left).

3.5.8 Instance Recognition on Different Objects Under Adverse Conditions

The previous sections have shown application of the filter cascade to produce improvements in several aspects of instance recognition for our running Humvee example. One question which arises naturally is whether we can generalize to other frames and other object instances. To test this, we generated a series of movies filled with a variety of different objects and ran those through the object classifier.

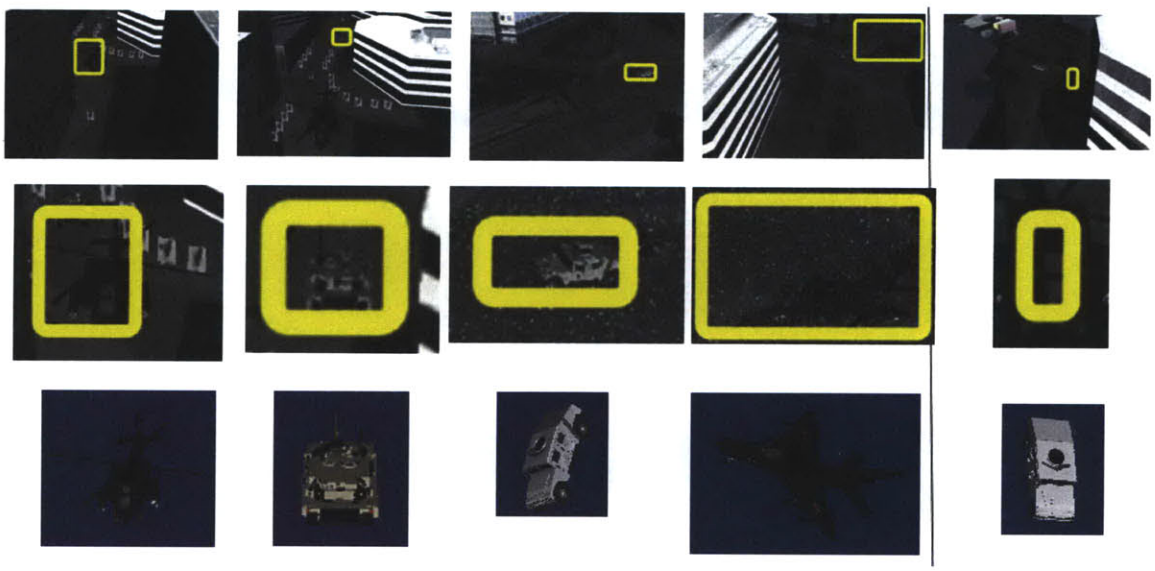

Figure 3-16: A few other instance recognition results showing input test images (top row), enlarged version of the detection area with bounding box (middle row), and the matching template in the database (bottom row). The majority are successes (left four) with some occasional failures (rightmost).

Figure 3-16 shows some examples of various object instances being identified and localized (shown with the bounding box). For the most part, the identification and localization works well. There are some occasional failures which are justifiable on the basis of similar appearance. For example, the false recognition of the cockpit of the Apache as a Humvee at a mostly top down orientation in Figure 3-16 can be explained by the fact that the both parts look like a grayish rectangle with a dark bar running through the middle.

3.5.9 Matching Real Images Against CAD Instances

Building on the previous section, a further generalization which is interesting to explore is to move towards real data. While the simulated video was made rendered in a reasonably realistic manner with noise and clutter, a useful system would need to be able to operate on real video. Given the instance recognition nature, we can really only expect to match real objects if they are similar in appearance to our model.

Development on this front shows some promise. One particularly interesting case

came up during an Internet image search for "aerial Humvee". The intent was to search for a picture of a Humvee taken from the air as the generated videos were designed for simulating an airborne sensor platform. The result instead was an airborne Humvee suspended underneath a plane. This is pictured in Figure 3-17.

Figure 3-17: A Google image search result for "aerial Humvee". Can we match this against our CAD model?

Can we detect this? It turns out that we can make a reasonable attempt at it. Figure 3-18 shows pure SIFT matches against a template Humvee and the matches after some "geometric grammar" filters were applied. While we were unable to eliminate all the false positives, we did eliminate a good number of them.

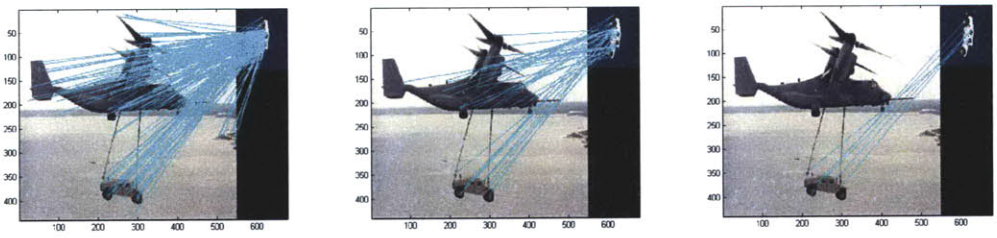

Figure 3-18: Results for SIFT feature matching between a real Humvee and our CAD model. "Geometric grammar" still helps in this case in cutting down on false positives.

Note that not only is our CAD model an inexact match, we also have confounding factors like very different intensity level for the background compared to that of the

template. This might be alleviated by comparing against a template generated with a lighter background. This would reduce the sharp intensity gradient along the object border which may affect the SIFT descriptors generated near the edge.

Overall, this direction is still very much a work in progress. There are certainly benefits to working with CAD models. However, finding models that are very similar in appearance to real objects may be difficult. Thus, it may be useful to generalize to investigating category recognition instead.

3.6 Category Recognition Evaluation

3.6.1 Experimental Setup

This experiment attempted to distinguish between airplanes and cars from the Caltech-101 dataset [13]. These are rigid body objects with fairly large numbers of training samples. In accordance with many other testing methodologies, 30 training samples were randomly selected from each class and the rest were used for testing. The training images were cropped to their bounding boxes while the entirety of every testing image was used.

3.6.2 Parameter Selection

SIFT features were extracted from the training images to form a visual vocabulary. There were approximately 25,000 total features in the training images. These features are then clustered using k-NN (nearest neighbor) clustering. Due to the randomized nature of k-NN, the results presented are the averages over 3 different runs of k-NN. A vocabulary size of 3000, corresponding to 3000 clusters, was chosen by experimentation as this value often produced clusters that look similar. An example of such a cluster can be seen in Figure 3-19.

The ROC curves in Figures 3-20 and 3-21 are generated with particular values of R (radius within which a pair of features is considered local and computations are performed on the pair) and the size of $W(z)$ (the number of words in the vocabulary

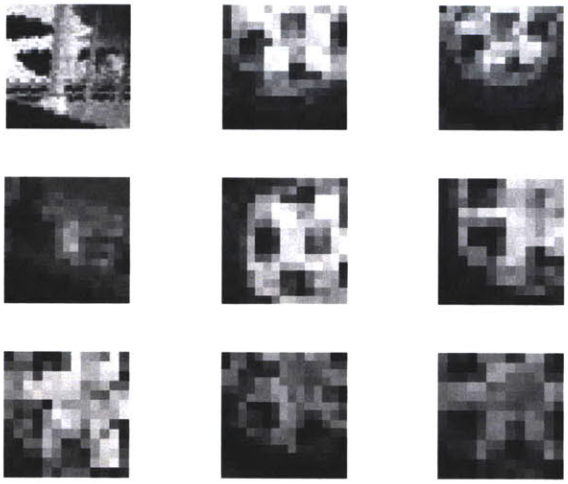

Figure 3-19: Reasonably successful vector quantization

Num Words	AUROC	Train	Test
30	70.23%	0.13s	33.3s
60	79.79%	0.47s	125s
90	86.42%	0.89s	225s
150	90.63%	2.19s	717s
250	91.39%	4.83s	1746s

Table 3.6: Area under ROC curve and running times for varying the number of words (in decreasing probability of occurrence) for which pairs are formed and consistency of the pairs checked according to the "geometric grammar" in the Caltech-101 airplanes class

out of which pairs are formed picked in descending order of probability according to the given class). Experimentation showed that a value of $R = 50$ worked well. Table 3.6 shows the results averaged over 3 runs for the airplanes class, varying the size of $W(z)$. 150 was chosen as the best trade-off in terms of speed and accuracy. Thus, for each category z, the set $W(z)$ is 5% of the w's (150 out of a total of 3000) sorted by $P(w|z)$.

3.6.3 Results

Figure 3-20 shows the ROC curves for the plane class (averaged over 3 runs) comparing a baseline "bag-of-words" implementation (based on that of Fei Fei *et al.* in [14]) to

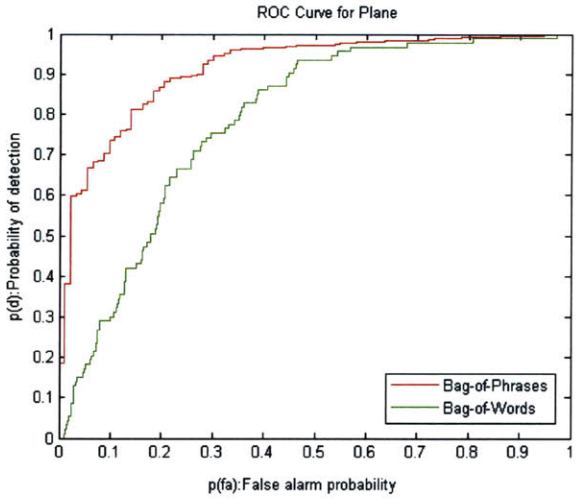

Figure 3-20: ROC curves for plane in Caltech-101. Note the improvement over baseline all along the ROC curve

the "bag-of-phrases" algorithm described in Section 3.4. Detection is improved across all false alarm threshold levels.

Figure 3-21 shows the ROC curves for the car class (averaged over 3 runs) making the same comparison. At low false alarm threshold levels, detection rate is improved. At high threshold levels, detection level is actually reduced. One possible explanation for the reduced detection is that there is great variability in the car class and we failed to model the boundary cases. While examining the 5% of the features that are most frequently occurring and exploring the geometric relationship between them allows for improved detection of prototypical cars, the less frequently occurring 95% of features may still be useful in encapsulating knowledge about boundary cases rather than just describing clutter. More research is necessary to determine how many of the features to use.

A comparison of classification accuracy rate is shown in Table 3.7. The "bag-of-phrases" approach has the highest classification rate in both classes. Results from another "bag-of-words" implementation [45] by Vedaldi is also included. While Vedaldi also trained on 30 training samples per class, further information on the testing methodology used was not disclosed. Thus, the results may not be directly comparable.

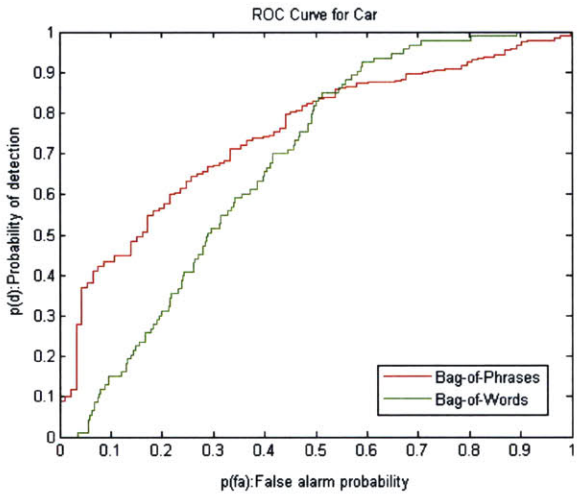

Figure 3-21: ROC curves for car in Caltech-101. Shows significant improvement in detection rate at low false alarm threshold but reduced detection at the high false alarm threshold.

Implementation	Plane	Car
Vedaldi bag-of-words	90.26%	79.57%
bag-of-words(pLSA)	90.99%	89.60%
bag-of-phrases	**92.12%**	**90.19%**

Table 3.7: Classification accuracy on Caltech-101 for planes and cars. Bag-of-phrases offers best accuracy in both cases.

3.7 Discussion

This chapter presented a "bag-of-phrases" model for object recognition. In Sections 3.5 and 3.6, this model was demonstrated to be superior to a "bag-of-words" model in both instance recognition and category recognition tasks.

In instance recognition, the proposed model is able to reduce the false positives by more than 2 orders of magnitude. The filter cascade portion of the "bag-of-phrases" model greatly improves robustness to noise, produces invariance to the SIFT matching threshold, and distinguishes between different orientations of an object. This is generalizable to several computer-generated object instances as well as some real world data.

In category recognition, even if the precise geometric arrangement is unknown, a "bag-of-phrases" model is still able to improve object detection rate when compared to a "bag-of-words" model.

Chapter 4

Tracking, Geolocation, and System Architecture

4.1 Overview

While object recognition goes a long way towards the goal of image and video understanding, there are benefits to adding other supplementary components which may improve the running speed and/or capability. In this chapter, we will explore the benefits of adding a tracker and a geolocation component as well as how to combine all three pieces together into a fast and effective system by utilizing mutual information. In Section 4.2, we will discuss adding in a tracker. In Section 4.3, we will discuss adding in a geolocation module. In Section 4.4, we will bring it all together in one combined architecture.

4.2 Tracker

4.2.1 Motivation

Why do we want to do feature tracking?

One reason is that tracking produces a way for us to obtain a consistent identity for an object over time. This can be very useful in understanding object motion for

applications such as video forensics and surveillance. A tracker can thus augment the object recognition capability described in Chapter 3.

Another reason is that tracking objects over time in video allows us to perform label transfer. Assuming that tracking is reliable, we are then able to generate object identity without needing to run object recognition over subsequent frames. Note here that the object recognition here can refer to either a human operator or an automatic computer algorithm. In the former case, we save an user/operator some attentional focus while in the latter, we save computational resources (trackers tend to be faster than object recognizers). Adding a tracker would thus confer savings of precious resources whether it is user attention or computational cycles.

4.2.2 Mutual Benefits from Integration

How A Tracker Benefits An Object Recognizer

As mentioned in the motivation portion, a tracker would primarily save time for the object recognition component. Given the current lack of teleportation technology, relatively smooth and continuous motion is a reasonable assumption even when both the object being observed and the sensor platform are in motion. Thus, knowing the identity and orientation of an object in a patch of one frame and reliably tracking to the next frame can produce very strong priors on the identity and orientation of the object in the new frame. The identity should be the same and the orientation should be either the same or change slightly. At the same time, the object recognizer only needs to process the area to which the object was tracked. This area should usually be a small portion of the entire frame. These two sets of information in conjunction dramatically reduce the search space for the object recognition algorithm and can thus produce a considerable speed benefit.

An example of the potential speed benefit is illustrated in Figure 4-1. This is the same running example from Section 3.5.3. With tracking into subsequent frames, the object recognizer only has to process a 120×60 patch rather than the full frame 640×480. Similarly, we do not have to consider all the possibilities of the object

being any of 4 object classes with 253 orientations each. Rather, given the previous identity of the object and orientation, we merely need to confirm its identity and which of the 9 similar orientations it is in the current frame. The 9 orientation are either no change from the last known orientation or up to 1 interval of difference in both the azimuth and elevation angles. The reduction in terms of frame area, object identity, and object orientation under consideration results in a roughly 3 orders of magnitude reduction in computational time for the object recognizer.

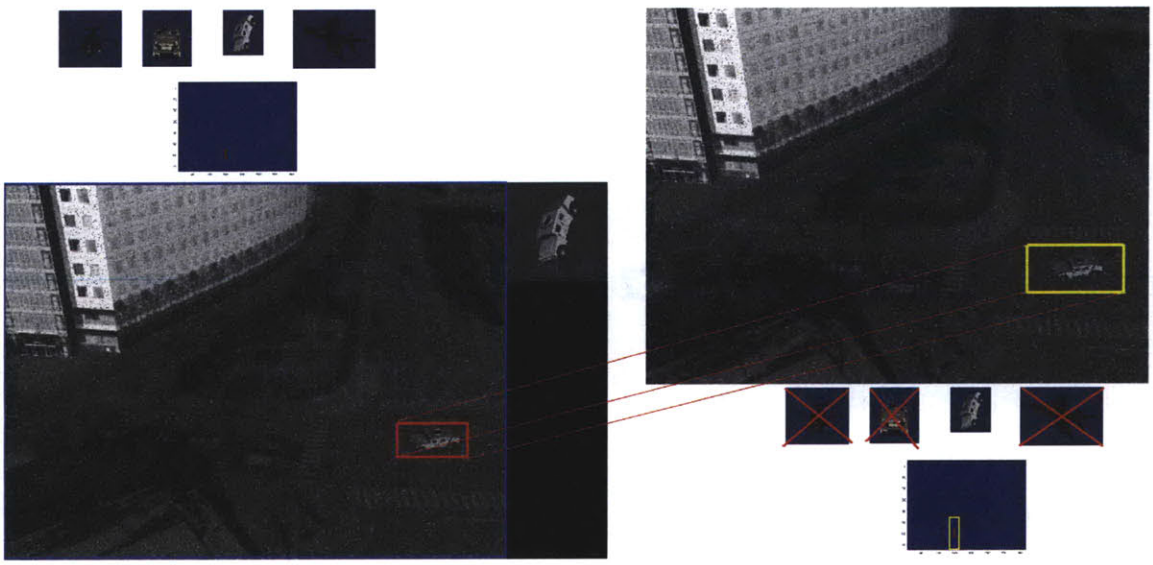

Figure 4-1: A tracker can benefit an object recognizer by greatly cutting down on the search space. The frame on the left shows the result of running the object recognizer over the entire frame (outlined in blue) and computing the scores for all object instances (the four shown in upper left) and for all orientations (represented as heat maps in left middle, all but the one for Humvee omitted for being uniformly zero which is shown as dark blue). After non-maximal suppression, only the orientation with highest score remains as a candidate(dark red). For the new frame, we need to only perform run the object recognizer on the portion of the new frame (boxed in yellow). At the same time, we only need to confirm that the identity is still that of a Humvee (only non-crossed out possibility in right middle) and that the possible set of orientations (outlined in yellow, right bottom) is similar to the last known orientation (dark red, right bottom)

How An Object Recognizer Benefits A Tracker

An object recognizer would be able to offer both a principled approach to updating the appearance model of the tracked object as well as some computational benefits for feature selection.

Trackers need to have a concept of what they are tracking. This occurs in the form of a template which represents the object being tracked. Due to the dynamic nature of video data, a tracker operating on a fixed template would usually rapidly lose track of the object. The tracker could suffer from compounding error and subsequently gradually drift off the object of interest if it is allowed to evolve the template. This issue was noted in [37]. Mathematically, the compounding mis-registration errors form a Markov Random Walk over time. The fundamental problem here lies in the fact that the tracker has no idea about what object is being tracked and thus is unable to evolve the template in a principled way. While some work has been done to model appearance in a principled way such as in the Fleet-Jepson tracker [24], these approaches still often include background distractors and do not run fast. Shi and Tomasi [37] use affine-evolving templates which can account for scale, rotation, and skew but not foreshortening or aspect changes. What we are proposing here is to go further and integrate a fast but relatively weak tracker with a full blown object recognizer which will be aware of the underlying object structure and thus produce accurate template updates. This prevents both drift, background distractors, and other problems which existing trackers encounter.

A combination tracker/object recognizer (top) performs significantly better than a tracker by itself (bottom) as shown in Figure 4-2. The addition of an object recognizer allows for principled template updates which effectively combats tracker drift and maintains a steady lock.

Computational Savings by Sharing Features

Sparse feature trackers tend to provide good tracking accuracy and speed. For this particular system, a pyramidal Lucas-Kanade tracker was chosen. A prerequisite for

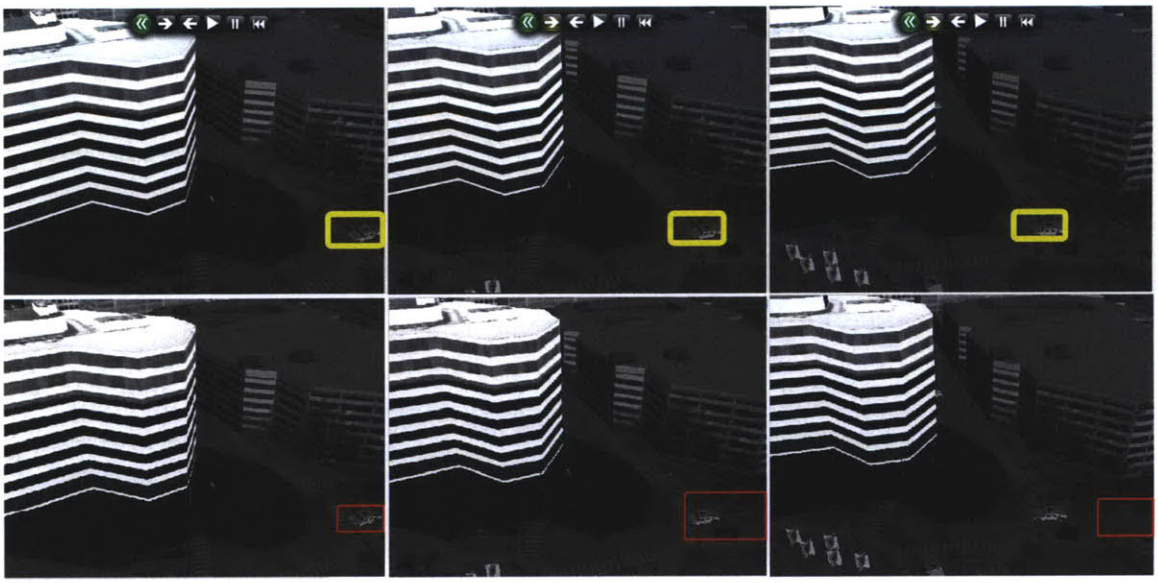

Figure 4-2: A steady lock is maintained with a combined tracker/object recognizer(top) while a tracker by itself (bottom) rapidly drifts off.

tracking is the selection of *good* features to track. The properties that make up a good feature to track is shown in the list below.

List of Good Features to Track

1. High saliency in the form of good gradient contrast

2. Stability in multiple gradient directions which leads to robustness under varying imaging conditions and noise

3. Semantic meaning relative to the object being tracked

Note that during the object recognizer portion, SIFT features which match the object are already extracted. These satisfy all the properties listed above. Passing the locations of these features to the tracker thus saves valuable time in performing feature selection and also aids in accuracy by ensuring that the features are a part of the object.

4.3 Geolocation

4.3.1 Motivation

Why do we want to perform geolocation?

One reason is to add a new capability. Geolocation of objects gives the ability for us to identify not only what an object is but also where the object is, which way it is facing, and even which way it is going. This is important for a deeper understanding of the video input. This does assume the existence of an accompanying telemetry stream to the input video stream. This telemetry stream would contain the positions and orientations of the camera for the duration of the video. While it may be possible to generate this telemetry stream from the video given suitable landmark recognition and vision-aided navigation capabilities, this work will assume the existence of a telemetry stream.

Another reason is to allow for tracking recovery/merging. If a track of an object is briefly lost, there is no way to tell if a new track belongs to the same object. With geolocation, however, it is possible to check the geolocated position of the object in the new track against the previously detected geolocation possibly taking into account motion priors. If there is a high confidence match, then the two tracks can be merged.

Yet another reason is to improve false positive rejection further. Note that for purposes like video forensics or surveillance, the ideal desired output would contain a single entry for each real object detected as well as offering the ability to display all the frames associated with said object. Geolocation enables us to cluster individual tracks into a single observation as desired. Sometimes, despite best efforts, there are some false positive frames. With geolocation, these frames would be unlikely to generate an observation with many frames while the true positive ones would. Hence, this allows us to filter out some of the false positives generated by the object recognition.

4.3.2 Mutual Benefits from Integration

How An Object Recognizer Benefits A Geolocation Module

An object recognizer provides the geolocation module with two important pieces of information.

The first piece of information is the location of the center of the object of interest in the scene. This point, in conjunction with the location of the camera, forms a line. Due to the fact that this line is acquired visually, it is typically called a "line-of-sight". A line-of-sight has the property that it goes through the object of interest. This fact can be exploited for purposes of geolocation. One popular method simply assumes that the object is on the ground and calculates the geolocation through the ground intersection of the line-of-sight. Another method, proposed by [33], finds the intersection of multiple lines-of-sight.

The second piece of information is the apparent scale of the object of interest in the image. This produces a prior on the distance from the camera to the object of interest. This prior frees us from the ground plane assumption.

These two pieces of information together provide a potential geolocation for the object of interest. The object should be in a "conic frustum of uncertainty" around this potential geolocation. An illustration of the "conic frustum of uncertainty" can be found in Figure 4-3. The conic frustum is generated by allowing for some error in both the pointing direction and the distance prior. Note that one way to reduce the pointing error is to define the center of the object of interest to represent the 2D projection of some canonical 3D point on the object. Nevertheless, some error may still exist and can be attributed to pixelization error, noise, or other factors.

How A Tracker Benefits A Geolocation Module

A tracker provides the geolocation module with additional information such as which lines-of-sight should be considered part of the same object. Thus, we can intersect many different "conic frusta of uncertainty" together and shrink down the amount of uncertainty. An illustration of this can be found in Figure 4-4. Section 4.3.3 presents

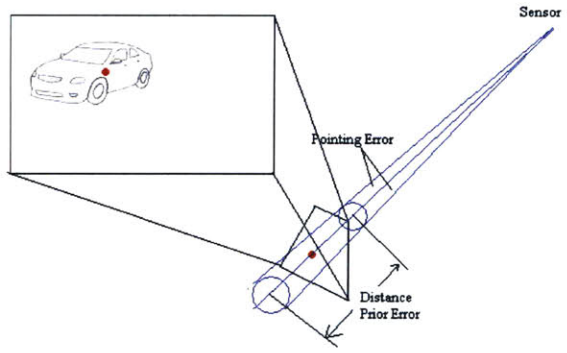

Figure 4-3: "Conic frustum of uncertainty" for geolocation from a single object recognizer output. A line can be drawn through the center of the object in the scene. Distance along that line can be estimated by the apparent size of the object. Both the direction of the line and the distance are subject to errors which results in the conic frustum.

several methods on performing the intersection.

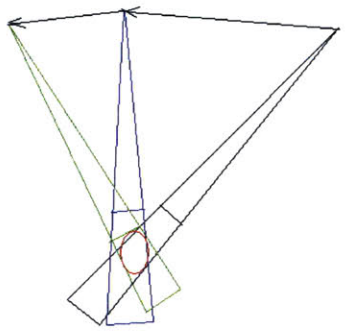

Figure 4-4: Tracking can provide multiple lines of sight to the same target (with corresponding "conic frusta of uncertainty"). This can great reduce the amount of uncertainty. Note here that the common region to all three "conic frusta of uncertainty" (circled in red) is significantly smaller than any given conic frustum.

How A Geolocation Module Benefits An Object Recognizer and A Tracker

At the same time, having a geolocation module will also yield benefits to an object recognizer and tracker. Despite best efforts, occasional errors such as false positive detections or data mis-association occur. The likelihood that these errors occur in a manner consistent with a geolocation point of view, however, is rather slim. Thus, the geolocation module will provide benefits in rejecting frames and/or tracks which

are unreliable.

4.3.3 Algorithm for Geolocation Based on Multiple Lines of Sight

With the object recognizer, we produce a line-of-sight for each frame detected. With the tracker, we can identify multiple lines-of-sight deriving from successive frames as belong to the same object. We will explore three different ways of producing geolocation estimates from these lines-of-sight: least squares, least squares with distance priors, and finally RANSACing these lines-of-sight. These three versions are presented in increasing order of complexity, reliability, as well as computational cost.

Least Squares

One naive method for performing geolocation is based upon the principle of least squares. In other words, we are attempting to find the point such that the sum of the squares of the distance from the various lines-of-sight to said point is minimized. Using least squares to perform this minimization results in the correct answer assumes that the measurements obtained are corrupted by zero mean Gaussian error. We assume the measurement error is of this form.

Let us consider first the case where n lines-of-sight intersect exactly at a particular point denoted as $(o_x, o_y, o_z)^T$. Similarly, let each line-of-sight be denoted by an offset (r_{ix}, r_{iy}, r_{iz}) and a slope (s_{ix}, s_{iy}, s_{iz}).

In this case, the equation $\mathbf{o} + a_i\mathbf{s_i} = \mathbf{r_i}$ is true for all $1 \leq i \leq n$.

Substituting in and expanding to matrix multiplication format $\mathbf{Ax} = \mathbf{b}$, yields Equation 4.1

$$\begin{pmatrix} 1 & 0 & 0 & s_{1x} & 0 & \cdots & 0 \\ 0 & 1 & 0 & s_{1y} & 0 & \cdots & 0 \\ 0 & 0 & 1 & s_{1z} & 0 & \cdots & 0 \\ 1 & 0 & 0 & 0 & s_{2x} & \cdots & 0 \\ 0 & 1 & 0 & 0 & s_{2y} & \cdots & 0 \\ 0 & 0 & 1 & 0 & s_{2z} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & 0 & 0 & \cdots & s_{nx} \\ 0 & 1 & 0 & 0 & 0 & \cdots & s_{ny} \\ 0 & 0 & 1 & 0 & 0 & \cdots & s_{nz} \end{pmatrix} \begin{pmatrix} o_x \\ o_y \\ o_z \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} r_{1x} \\ r_{1y} \\ r_{1z} \\ r_{2x} \\ r_{2y} \\ r_{2z} \\ \vdots \\ r_{nx} \\ r_{ny} \\ r_{nz} \end{pmatrix} \tag{4.1}$$

By inspection, the matrix $\mathbf{A}$ has linearly independent columns and is thus rank $n+3$ in the general case. A least squares solution can be calculated by the standard formula of $\mathbf{x} = (\mathbf{A^T A})^{-1}\mathbf{A^T b}$. Observe that $A^T A$ is invertible as it is full rank $(n+3)$. Note that rank of $n+3$ only occurs if there are two or more measurements. If there is only a single measurement, just form the estimate via traversing a distance of prior down the line-of-sight.

Equation 4.1 provides the formulation for all stationary targets. It is, in fact, possible to modify this equation to account for arbitrary parametric motion.

Below, a simple example for the linear, constant velocity case is presented. In this case, the target moves at constant velocity $\mathbf{v}$ starting from offset $\mathbf{o}$ at $t = 0$. A line-of-sight formed by $\mathbf{r_i}$ and $\mathbf{s_i}$ models the motion at $t = t_i$.

In this case, the equation becomes $\mathbf{o} + \mathbf{v}t + a_i\mathbf{s_i} = \mathbf{r_i}$. Writing out all the equations in matrix format yields Equation 4.2.

$$\begin{pmatrix} 1 & 0 & 0 & t_1 & 0 & 0 & s_{1x} & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & t_1 & 0 & s_{1y} & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & t_1 & s_{1z} & 0 & \cdots & 0 \\ 1 & 0 & 0 & t_2 & 0 & 0 & 0 & s_{2x} & \cdots & 0 \\ 0 & 1 & 0 & 0 & t_2 & 0 & 0 & s_{2y} & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & t_2 & 0 & s_{2z} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & & & \\ 1 & 0 & 0 & t_n & 0 & 0 & 0 & 0 & \cdots & s_{nx} \\ 0 & 1 & 0 & 0 & t_n & 0 & 0 & 0 & \cdots & s_{ny} \\ 0 & 0 & 1 & 0 & 0 & t_n & 0 & 0 & \cdots & s_{nz} \end{pmatrix} \begin{pmatrix} o_x \\ o_y \\ o_z \\ v_x \\ v_y \\ v_z \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} r_{1x} \\ r_{1y} \\ r_{1z} \\ r_{2x} \\ r_{2y} \\ r_{2z} \\ \vdots \\ r_{nx} \\ r_{ny} \\ r_{nz} \end{pmatrix} \tag{4.2}$$

While the least squares solution is very general, there are some problems with robustness. In particular, least squares is notoriously sensitive to outliers. Furthermore, even if only a few lines-of-sight are obtained in succession before the target is lost due to noise, occlusion, or other factors, then the narrow baseline results in a fairly substantial depth ambiguity. A substantially wrong depth would result in unsuccessful tracking recovery through geolocation. Even worse, it could result in data association errors when such an attempt is made.

While some constraints can be imposed on the solution such as prohibiting the least squares solution to lie behind the camera on any given line-of-sight, there is little that can be done when such a failure is detected.

Least Squares with Distance Priors

One method to compensate for the brittleness of the least squares intersection method utilizes the distance prior that we have obtained via the apparent scale of the object. By forcing the least-squares solution to be at most $IDENTITY\_RADIUS$ away from each line-of-sight and deviate by a multiplicative distance factor of at most $DISTANCE\_PRIOR\_ALLOWANCE$ away from the estimated distance prior, we are forcing the least squares solution to lie within the "conic frustum of uncertainty" for any given line-of-sight. This provides us a robust way to reject least-squares

solutions that are implausible such as a car floating high off the ground.

Unfortunately, even if we reject a solution, there is no indication as to which line-of-sight is the outlier. Often, we encounter the case of narrow baseline matching where the depth can be very uncertain. One approach to handle this is by maintaining an invariant such that each set of lines-of-sight must have a least-squares solution which lies within the "conic frustum of uncertainty" for any given line-of-sight. If adding another line-of-sight would result in the invariant not holding, then said line-of-sight will be split into a new track.

While this approach is fairly effective at ensuring that only frames corresponding to the same object are grouped together, there are certain issues. One such issue occurs when a frame produces a measurement that drifts only a little or results in a small mis-association error. This thus violates our underlying single object with Gaussian error assumption. This may not be detected right away as an outlier. However, as additional measurements arrive, the invariant may be violated due to it being an actual outlier. The proposed method would reject the newest measurement and start a new track. However, the correct thing would be to actually examine all the measurements and reject the previously undetected outlier. As a result, this method may end up breaking a single object into many different tracks. This is problematic as we would like to have a single track for each object. The remedy to this problem is presented in the next section.

RANSAC Lines-of-Sight

One way to incorporate a mechanism to accommodate earlier mistakes is via the use of a RANSAC-like approach to reject outliers. If the least-squares solutions for a set of lines-of-sight does not satisfy the distance priors, we will attempt to find the maximal consensus set which does satisfy the priors and throw out the outliers. In this case, the outliers need not be the newest lines-of-sight added.

In each iteration, we will randomly select two lines-of-sight and find the least-squares intersection of these two lines. If the intersection lies within the "conic frustum of uncertainty" for both, then the consensus set is built by observing the number

of other "conic frusta of uncertainty" that this intersection lie in. If this consensus set is larger than the previous best consensus set, then it is saved as the best consensus set thus far. Run this procedure for many iterations and we have a good idea of which lines-of-sight are inliers and which are outliers.

This method enables rejection of outlier line-of-sight measurements from any time during a track and is hence the most robust out of the three we examined. However, the computation time from running many iterations of RANSAC can be significant. Least-squares with distance priors can function as a reasonable alternative if computation time is a major concern. In our system, we utilize RANSAC Lines-of-Sight as the geolocation method of choice.

4.4 System Architecture

As we have seen in the preceding sections, there are many benefits to integrating the three components of object recognizer, tracker, and geolocation module into a single system. These components provide interlocking constraints which mutually improve performance, making the system more than a sum of its parts. The proposed architecture is presented in Figure 4-5.

In the architecture presented in Figure 4-5, state of the system is encapsulated by a data structure which consists of a list of detected objects. This list encapsulates knowledge of all the objects that have been ever detected by this system. This list is subdivided into active and inactive categories. This refers to whether or not the system is actively tracking these objects and thus expecting to see them in the next frame.

When a new frame comes into the system, it is fed to the tracker which tracks all the active objects from the previous frame to the new frame and outputs the corresponding bounding boxes. Here the bounding boxes are expanded by a multiplicative factor $DILATION\_FACTOR$ to allow for capturing the immediate neighborhood of the object in addition to the tracked region. The immediate neighborhood might be used in generating features. Only the portions of the frame inside those bounding

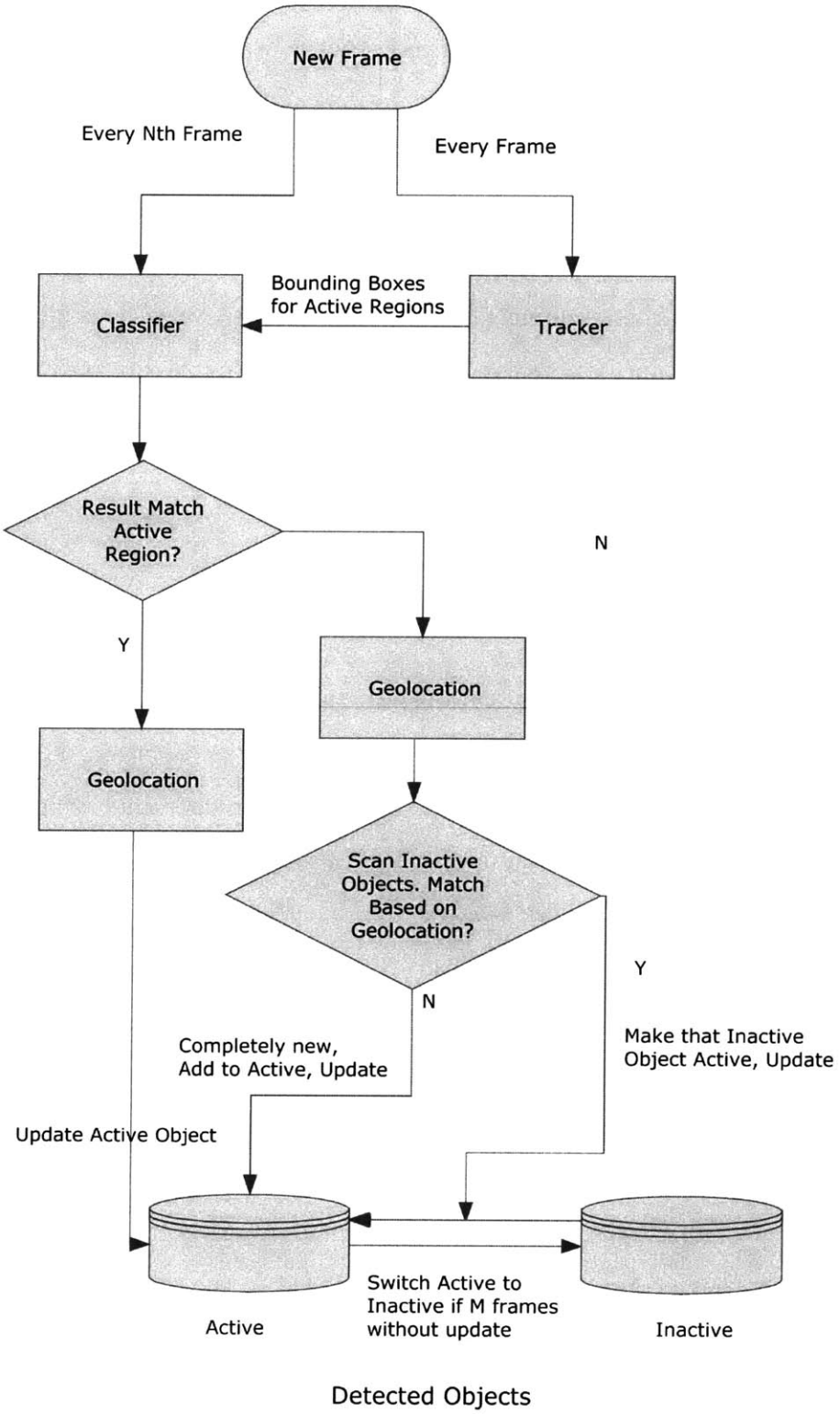

Figure 4-5: System Architecture Diagram

boxes are subsequently sent to the classifier to confirm the identity in the new frame. This is a performance optimization which greatly cuts down on the search space in both image space and object/orientation space as mentioned in Section 4.2.2. Every N^{th} frame, full frame object recognition allows us to detect new objects. This also provides an opportunity to recover previously found objects which we lost track of.

Regardless of whether the frame is scanned completely or just the regions indicated by the tracker, the results are segmented. Each result that is associated with an actively tracked region go on to extend that active track which is updated correspondingly. For the other results, they are checked against the list of inactive detected objects based upon both geolocation and identity. Note that here geolocation can either refer to a static location or a predicted location based upon an assumed motion model. If it matches an inactive object, then the inactive object is made active again and updated with the new frame. If a given result does not match any of the previously detected objects (active or inactive), then it is considered to be a new object and added to the active list.

At the end of this process, a scan is made through the active list. If an object on the active list has not been updated for M frames, then it is moved to the inactive list. With the intervening motion, the template is most likely very out-of-date. Chances of matching an out-of-date template to the same object in the new frame is minimal. Thus, it is best to remove the object from the list to save on computation even if doing so potentially comes at a slight cost for accuracy (that scenario occurs if the tracker would have worked if allowed to run and the recovery mechanism described makes a data association error upon reactivation of the track).

Note that another option is to generate a new predicted template based upon the intervening predicted motion of the object as well as the known motion of the sensor and then use the new template to perform matching. However, this comes at the cost of computation. Many times, the reason for losing track is due to occlusion of the object. Even the new template will not help in that scenario. Overall, it may be advisable to simply give up on the track as lost and attempt to pick it up again during the next full frame object recognition phase.

4.5 Discussion

This chapter presented the rationale and benefits to integrating an object recognizer, a tracker, and a geolocation module. Furthermore, it presented an architecture which combine these components together in an interlocked whole, enabling communication for mutual improvement through interlocking constraints.

Chapter 5

Practical Video Understanding With VICTORIOUS

VICTORIOUS is a fast and robust implementation of the architecture described in Chapter 4. This implementation serves as a proof of concept for combining a "bag-of-phrases" instance recognizer, a tracker, and a geolocation module into a practical system. This implementation can be used to automatically process and index multiple videos to save time for a human operator. In this chapter, Section 5.1 describes the design philosophy, Section 5.2 showcases several components of the implementation, and Section 5.3 provides an experimental evaluation of VICTORIOUS.

5.1 Design Considerations

The first design consideration and primary goal of VICTORIOUS is to provide an automatic video indexing tool to perform "video triage" for human operators who lack the time to pore through large amounts of video manually. "Video triage" works by ranking the video data to place the most important portions at the top. The envisioned usage case includes feeding the output of this system to a GUI such as the one shown in Figure 5-1 to facilitate human interaction and decision making. The human interaction aspect means that some small errors in the results can be tolerated and overcome by the user. However, for user satisfaction and efficiency

reasons, *accuracy* is still important. In many scenarios, the user would like to not miss anything, even if that means sorting through a few false positives. In document retrieval terminology, 100% recall is very much desired even at the cost of some precision.

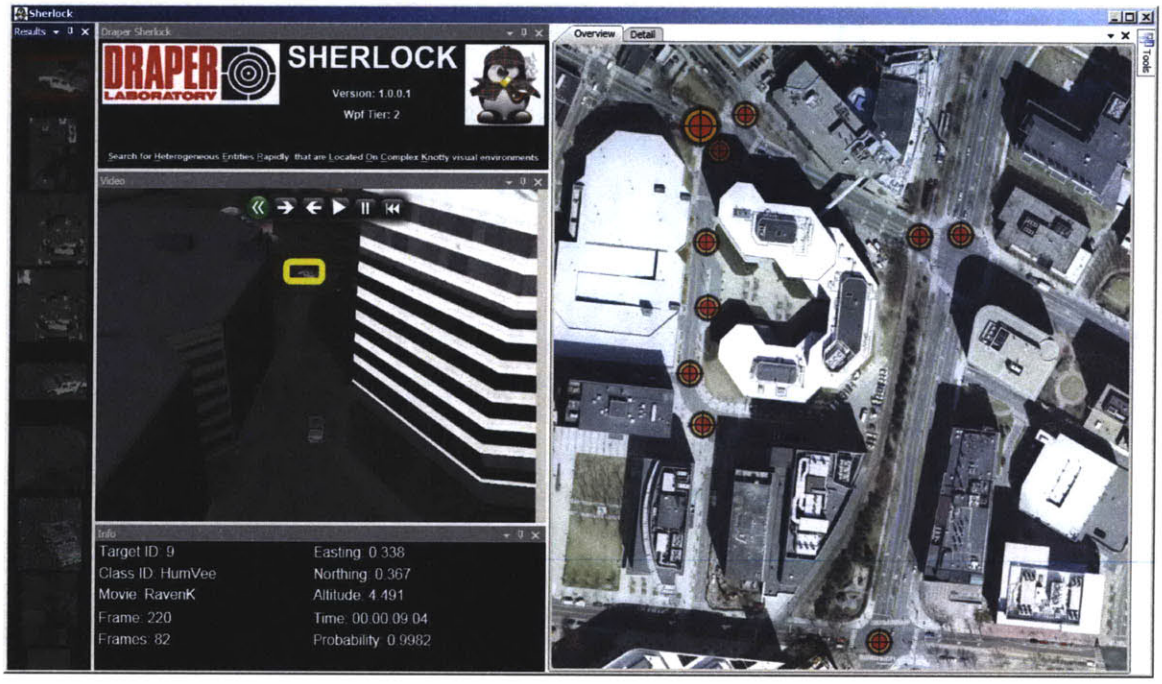

Figure 5-1: Example of a GUI (Pictured SHERLOCK, courtesy of Yechezkal Gutfreund) for displaying the results of VICTORIOUS. Some of the information presented may include a ranked list of the objects of interest found (left panel), video controls with the relevant portions of the video highlighted (center panel), and an overhead map showing geolocation of the objects (right panel). The top ranked detection (out of many) is highlighted and shows a tight bounding box in the video as well as accurate geolocation on the map.

The second design consideration which is related is *robustness*. Given wide variations in data sources and conditions, the system must perform at or near the same level for all of them.

The third design consideration to be traded off against is *speed* (along with the associated element of *scalability*). Although this prototype system VICTORIOUS is being built for an offline scenario, running at high speed can still free up computational resources for other tasks. Furthermore, high-speed operation would enable moving to real-time applications such as navigation. Scalability would allow moving

to a multiple sensors/multiple processors system which can further improve speed, accuracy or both.

5.2 Implementation of VICTORIOUS

5.2.1 Instance Recognizer

The instance recognizer takes advantage of several sets of binaries listed below. David Lowe's SIFT demo code [30], which is C++ code with a Matlab wrapper, performed SIFT feature extraction. I wrote a Matlab wrapper around Changchang Wu's SiftGPU code [47] and used that to do SIFT feature matching. I implemented the filter cascade portion in C++. Marco Zuliani's RANSAC Toolbox [50], which is implemented in Matlab, ran MLESAC. I wrote a framework in Matlab that combines all of these elements together into the instance recognizer.

5.2.2 Tracker, Geolocation, and Integrated Architecture

The OpenCV [1] Pyramidal Lucas-Kanade sparse feature tracker (C++) with a Matlab wrapper performed the feature tracking. I implemented the RANSAC LOS code in Matlab that performed the geolocation functionality. I wrote a full framework in Matlab that implemented the architecture described in Section 4.4 to connect and integrate all of the components for testing and demonstration.

5.3 Evaluation

5.3.1 Experimental Setup

As mentioned before in Section 3.5.1, four CAD models of vehicles were downloaded from the web. These consisted of an Abrams tank, an Apache helicopter, a Humvee, and a MiG jet. A CAD model of the Technology Square area was also obtained.

An application was developed based upon the OpenSceneGraph library (an Open Source object rendering/simulation framework built on top of OpenGL) to read in

CAD models and render images containing them from arbitrary relative positions and orientations. This application was then used to generate both templates and test data.

The templates were generated by taking snapshots of the CAD vehicles at 10^o intervals of out-of-plane rotation against an uniformly colored background. The test data was generated by placing the CAD models into the Tech Square model and then flying the camera around, rendering from different position and orientations.

A total of 7 movies were generated that contain varying levels of salt-and-pepper noise (from none to up to 15% area coverage) as well as varying camera paths (straight, straight with shaking, circling, etc.). This is meant to model a variety of possible imaging conditions deriving from CCD defects and/or sensor conditions (an airborne robot may be asked to circle a particular object, just fly straight, encounter air turbulence, etc.)

The goal in this system test is to consider whether or not the system as a whole can perform the task of object detection, tracking, and geolocation in an accurate and robust manner.

5.3.2 Research Questions

There are a number of research questions which we would like to answer through experiments in the following sections.

1. There are a lot of parameters in this system. How robust are we to variations in those parameters?

2. How accurately can the object recognition detect the objects of interest? We will treat this as an information retrieval problem and judge how well we accomplished the task of retrieving all the relevant objects.

3. How well does tracking work? What percentage of frames that we detect do we group into single sightings?

4. How well does geolocation work?

5. How fast does this run? Can we run this system in real time?

5.3.3 Parameter Sensitivity

In the results for the upcoming sections, the default parameter setting and its 28 variations (14 that vary one up and 14 that vary one down) are presented unless otherwise noted. While this process does not perform an exhaustive search of the parameter space, it does provide some intuition about the level of sensitivity of the overall system to any single parameter. As a whole, this system is relatively insensitive to the parameters.

Furthermore, a preliminary run using the default parameters showed that performance is reasonably similar across the movies. 3 of the movies were picked as representative and all the parameters runs were performed on these three movies for the sake of running time.

5.3.4 Accuracy of Object Detection

For the user scenario described in the first design consideration, we hope to find all the relevant results while including as few false positives as possible. One way to visualize performance is via a recall-precision curve. The 3 movies together contained 13 objects. The recall-precision performance for the 29 different parameter settings is shown in Figure 5-2.

From Figure 5-2, we can see that the recall-precision performance is largely consistent across parameter variations. We achieve the goal of 100% recall without many false positives in 26 out of 29 parameter settings. We are able to achieve 100% recall and precision for 8 out of 29 settings which is a substantial fraction of the total. For 20 out of 29 settings, there are one or fewer false positives.

5.3.5 Tracking Reliability

It is also helpful to characterize the tracking performance. Tracking performance can be investigated by how frequently frames are dropped in the middle of a set of frames

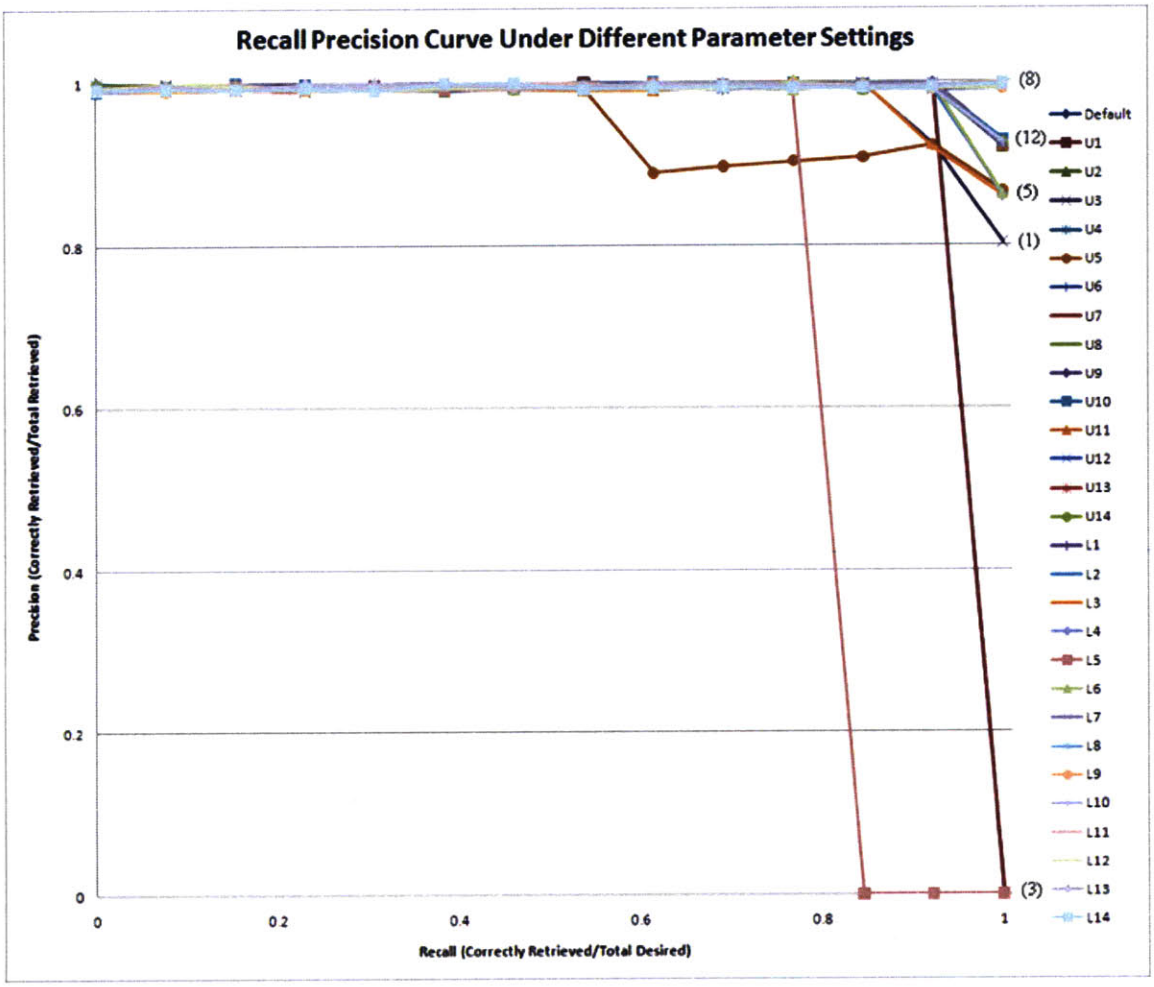

Figure 5-2: Accuracy is insensitive to parameter variations. Recall-precision curves show that we achieve 100% recall with greater than 80% precision in 26 out of 29 parameter settings. Further note that for 8 of the parameter settings, we achieve 100% precision while for 12 of them, we achieve 93% precision corresponding to only one false positive.

detected as belong to a particular object. If the tracker has poor reliability, then more frames will be dropped in the middle. Thus, one measure of tracker reliability is the density of the recovered frames (within a continuous set of frames where an object is visible, the fraction of those frames where the tracker detects the object). Figure 5-3 shows a plot of tracking densities for all 3 movies for all parameter settings.

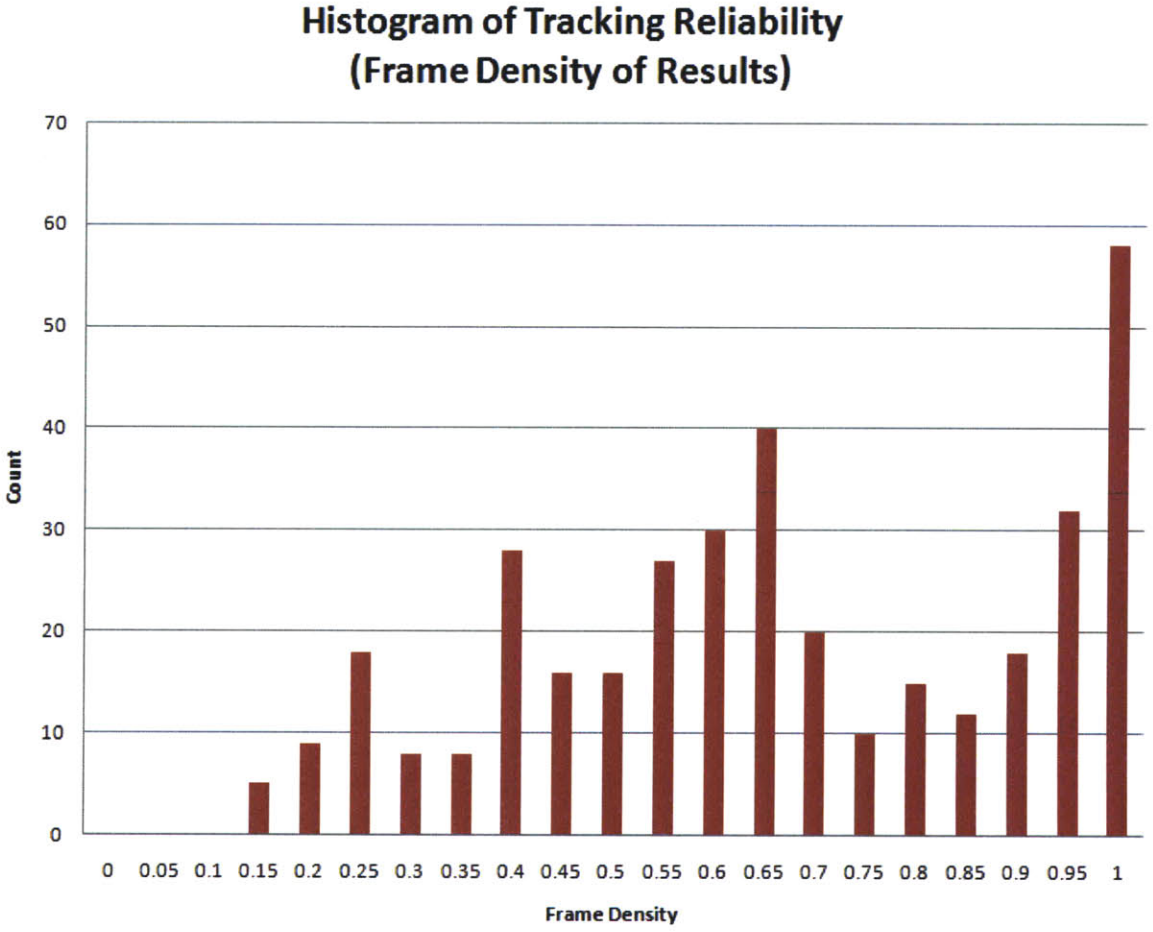

Figure 5-3: Frame detection density (more is better) for various objects of interest, movies, and parameter settings. This is a measure of the reliability of tracking. Perfect tracking would result in a frame detection density of 1. The 25^{th}, 50^{th}, and 75^{th} percentiles are 0.452, 0.634, and 0.896 respectively.

In Figure 5-3, we can see that the frame densities are reasonably high. More than half of the frame densities exceed 0.6. This degree of tracking performance as measured by recovered frame density results in fairly smooth playback of the video frames associated with each detected object. This level of smoothness will likely be

considered acceptable to the user by not causing significant skipping. Of course, there is certainly still significant room for improvement in terms of tracking reliability. In particular, some values can be as low as approximately 0.1.

Overall, this tracker density measure is not sensitive to parameter variations. Rather, most of the density variations come from different detected objects. Certain objects with relatively few features tend to cause low frame densities.

5.3.6 Geolocation Accuracy

Geolocation accuracy is yet another important aspect of the system's performance. Again, geolocation accuracy appears to be robust to parameter variations. The predominant source of difference seems to derive from the class of the object being detected. Figure 5-4 shows the distributions for geolocation error based upon class. For reference, the camera in each video is at least 100m away from the objects of interest.

From Figure 5-4, the geolocation errors for all the classes appear to be largely below 2 meters with the exception of the Apache. Thus, it can be useful to examine the Apache vs. all the other classes. The geolocation errors for this comparison are plotted in Figure 5-5.

From Figure 5-5, we can see that the geolocation error for classes other than Apache is fairly good as more than 90% of the measurements have 1.88m of error or less. While the geolocation errors for the Apache class are worse, more than half of the measurements still have less than 2m of error.

The difference in geolocation accuracy can be explained by the fact that the Apache has rotors that are thin and long, making the outline of the Apache more concave. Features on rotors tend to be unreliable due to the fact that they also capture a significant amount of background distractors. This is less of a problem for the other objects which are more convex.

In the histogram of the geolocation error for all classes excluding Apaches, there appear to be two spikes. These spikes correspond to two groups of objects. The group with less geolocation error contains objects which are close to the camera and

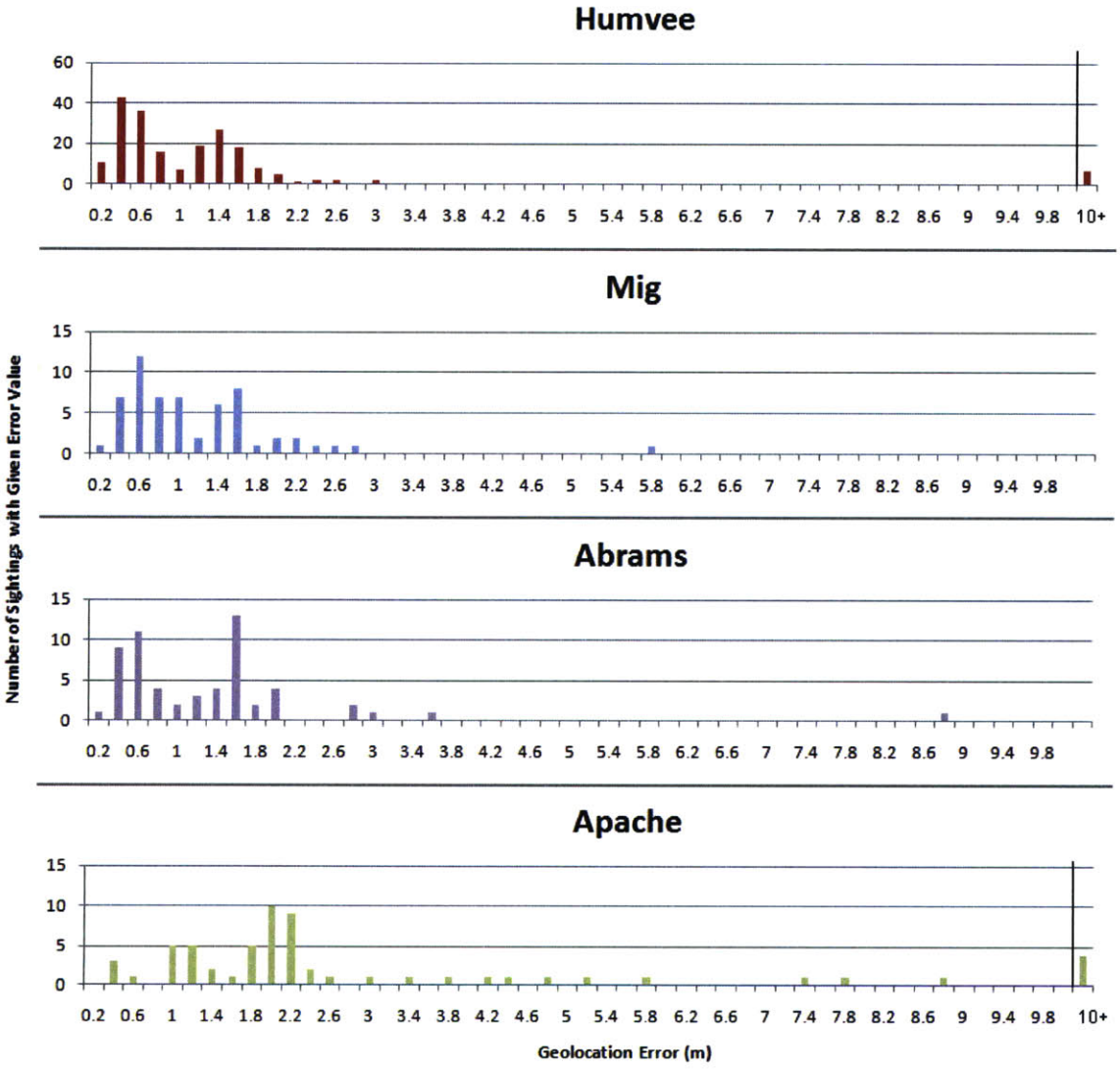

Figure 5-4: Geolocation errors for each class as a distribution collated over all the movies and parameter variations. Each distribution is normalized. Note here that Apaches seem to perform worse while others are fairly similar.

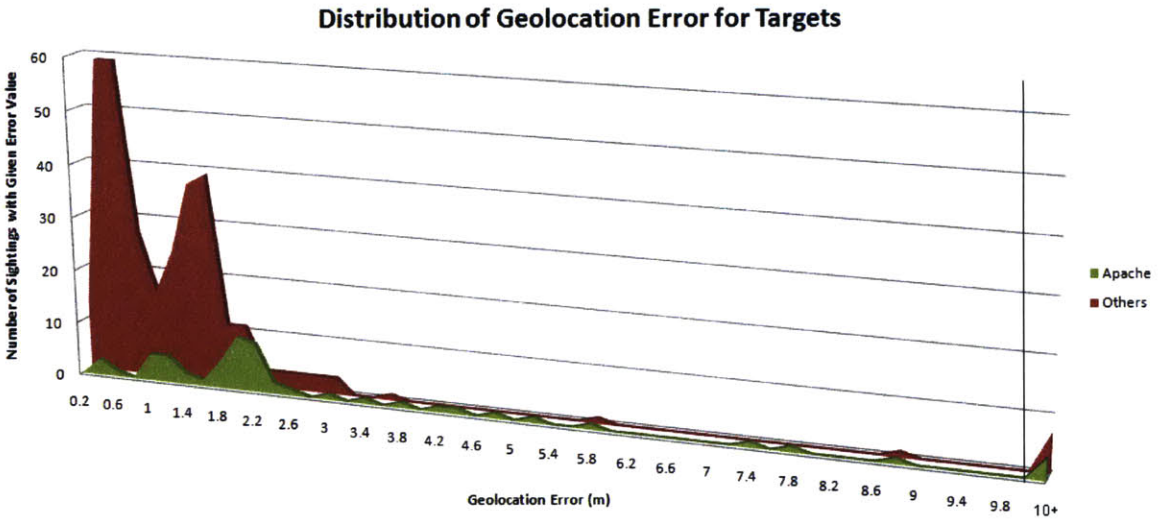

Figure 5-5: Geolocation error for Apache and all other classes. This is the same data as Figure 5-4 except the Humvee, Abrams, and Mig classes are merged together and the vertical axis is counts rather than normalized frequency. The 0.5, 0.75, 0.9 CEP cutoffs for Apaches are 1.9m, 2.62m, and 7.32m respectively. The 0.5, 0.75, 0.9 CEP cutoffs for all other classes are 0.8m, 1.42m, and 1.88m respectively.

relatively unaffected by noise. The group with more geolocation error contains objects which are either far from the camera and/or are obscured by noise. The difference in number of reliable features extracted for the objects between the groups results in the difference in geolocation error.

Note that in all these cases, the telemetry stream being used is perfectly accurate. Any error derives from pixelization effects, compression artifacts, or failures in the object recognizer. Operating on real world telemetry may result in increased geolocation error.

5.3.7 Runtime

For many applications, real-time performance is desired. A system which can search for and detect 25 different models at 30Hz would be very useful. In this section, we explore the possibility of meeting this goal. The parameters, with the exception of the SIFT match threshold, do not affect the running time of the system to any significant extent. The first column of Table 5.1 shows a representative estimate of the amount

of time taken in each part of the system. The current hardware used for this timing includes a 3.0GHz dual-core processor with 2 GB of memory and a mid-range GPU (nVidia 9600GSO^{TM}). Subsequent columns in Table 5.1 show the improvements that are predicted with porting all the code over to the GPU, switching to 15^o intervals for the orientation spacing rather than the current 10^o intervals, and finally changing over to a quad Tesla^{TM}(a GPU computation card) system.

	Current	GPU	15^o Interval	4x Tesla
SIFT Extraction	1.6678(C++)	0.04	0.04	0.0066
SIFT Match (ea. model)	0.0274(GPU)	0.0274	0.0122	0.0005
Constraint (ea. model)	0.1797 (C++)	0.0019	0.0008	0.000033
RANSAC (ea. model)	0.0795(MAT)	0.0007	0.0007	0.000029
Track (ea. model)	0.0986 (C++)	0.0099	0.0099	0.0004
Rest (ea. model)	0.0011(MAT)	0.0001	0.0001	0.000004
Total Time (25 models)	11.3253s	1.04s	0.6325s	0.0308s

Table 5.1: Running speed of current system and potential for real time performance (25 models, 30Hz) with a more optimized implementation on GPU.

Some preliminary experimentation indicate that the numbers quotes for porting over to the GPU are credible. Indeed, the estimate may be conservative based upon the time savings that would result from not having to transfer data between the GPU and CPU across the system bus.

The change from 15^o to 10^o intervals should reduce the SIFT matching and constraint phases by a factor of 4/9 ($\frac{10}{15}^2$ as the intervals apply for both the azimuth and elevation angles). A further improvement could be obtained by using wider intervals near the poles or other ideas for arrangement of the orientations. The move away from a grid-like arrangement would necessitate introduction of an auxiliary adjacency data structure. The overall improvement in run time would justify this increase in complexity. Also, the switch from a single mid-range GPU to a quad Tesla^{TM}system should result in a roughly 25x performance boost as each Tesla ^{TM}has roughly 6 times the computational power of the GPU currently used.

5.4 Discussion

This chapter presented some details on the implementation VICTORIOUS which serves as a proof of concept of the algorithms and system architecture discussed in the preceding chapters. This chapter also characterized the performance of this system in terms of parameter sensitivity, object recognition accuracy, tracking reliability, geolocation accuracy, and running time. The system presented is found to be insensitive to parameter variations for the most part. Object recognition resulted in perfect recall and greater than 80% precision in 26/29 cases. Tracking reliability is also fairly high. Geolocation is accurate to within 2m the vast majority of the time despite over 100m of distance from sensor to the objects of interest. Running time is also fast with good possibility of achieving real time performance (25 models at 30 Hz) using current generation hardware.

Chapter 6

Conclusion

In this thesis I have described some of the requirements of a video understanding system. I have also shown that although many existing algorithms can accomplish the separate component tasks (object recognition, tracking, and geolocation) accurately, they may not be as robust or as speedy as desired. The main contributions of this thesis are a "bag-of-phrases" model for object recognition, a proposed architecture for combining modules that perform these individual tasks (enabling mutual complementarity), and a prototype implementation based upon this architecture.

I have introduced a "bag-of-phrases" model, which extends the standard "bag-of-words" model of object recognition by developing a "geometric grammar" to check for consistency. A cascade application of the geometric grammar rules is computationally efficient and highly effective. I evaluated this model on several datasets and demonstrated that this new model significantly increases accuracy and robustness in both instance and category recognition tasks compared to a "bag-of-words" baseline. The instance recognition variant of this model is able to achieve more than two orders of magnitude reduction in the number of false positives while eliminating virtually no true positives.

I have illustrated the benefits that each of the three components can bring to each other through interlocking constraints. These powerful interlocking constraints combine to make an architecture which is more than the sum of its parts.

I have developed VICTORIOUS as a practical demonstration of the accuracy, ro-

bustness, and speed of the architecture. This system proved to be robust to parameter variations and is able to achieve 100% recall and >90% precision, <2m of geolocation error, and >0.6 tracking reliability for the majority of the parameters. Speed experiments and predicted implementation changes suggest that real time performance is possible on current generation hardware with no algorithmic changes necessary.

6.1 Recommendation for Future Work

While this thesis has laid some groundwork for practical video understanding systems, much work remains to be done. While the current system is able to handle instance recognition in a robust and speedy way, more work must be done to allow for category recognition to perform at the same level. Learning features to distinguish between categories (car) and subcategories (sedan) in an object hierarchy will be another promising area of research. These learned features can provide a better user experience by allowing a system to explain to users their rationale for making a particular decision. This additional interactivity would make failures justifiable to the user as well as allow for the possibility of learning from mistakes made by the system and corrected by the user.

In addition to the algorithmic challenges involved with building a real time, practical video understanding system, we also face many implementation and design problems. Experimentation with GPUs suggest a need for careful algorithm design and parallelization as well as concern for bandwidth management in order to achieve real-time performance.

Bibliography

[1] OpenCV. `http://sourceforge.net/projects/opencvlibrary/`, 2009.

[2] Padmanabhan Anandan. *Measuring visual motion from image sequences.* PhD thesis, 1987.

[3] A. P. Ashbrook, N. A. Thacker, P. I. Rockett, and C. I. Brown. Robust recognition of scaled shapes using pairwise geometric histograms. In *BMVC '95: Proceedings of the 6th British conference on Machine vision (Vol. 2)*, pages 503–512, Surrey, UK, 1995. BMVA Press.

[4] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12:43–77, 1994.

[5] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, 2002.

[6] A.C. Berg, T.L. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondences. In *Computer Vision and Pattern Recognition*, volume 1, pages 26–33 vol. 1, 2005.

[7] Gertjan J. Burghouts and Jan-Mark Geusebroek. Performance evaluation of local colour invariants. *Comput. Vis. Image Underst.*, 113(1):48–62, 2009.

[8] Marco La Cascia, Stan Sclaroff, and Vassilis Athitsos. Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. *In IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:322–336, 1999.

[9] Kar Wee Chia, Adrian David Cheok, and Simon J. D. Prince. Online 6 DOF augmented reality registration from natural features. In *In Proceedings of the International Symposium on Mixed and Augmented Reality*, pages 305–313, 2002.

[10] Gabriela Csurka, Chris Dance, Jutta Willamowski, Lixin Fan, and Cedric Bray. Visual categorization with bags of keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision*, 2004.

[11] Department Of Engineering, B. Tordoff, W. W. Mayol, T. E. De Campos, and D. W. Murray. Head pose estimation for wearable robot control. In *In Proc British Machine Vision Conference*, pages 2–5, 2002.

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. Pascal VOC2008 results. `http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2008/results/index.shtml`, 2008.

[13] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *Computer Vision and Pattern Recognition Workshop*, page 178, 2004.

[14] L. Fei-Fei, R. Fergus, and A. Torralba. ICCV 2005 short course: Recognizing and learning object categories. In *International Conference on Computer Vision*, 2005.

[15] R. Fergus, L. Fei-Fei, P. Perona, and A. Zisserman. Learning object categories from Google's image search. In *International Conference on Computer Vision*, 2005.

[16] David J. Fleet and A. D. Jepson. Computation of component image velocity from local phase information. *Int. J. Comput. Vision*, 5(1):77–104, 1990.

[17] W. Eric L. Grimson and Tomas Lozano-Perez. Model-based recognition and localization from sparse range or tactile data. *Readings in computer vision: issues, problems, principles, and paradigms*, pages 382–414, 1987.

[18] W.E.L. Grimson, T. Lozano-Perez, and D.P. Huttenlocher. *Object Recognition by Computer: The Role of Geometric Constraints*. MIT Press, 1990.

[19] G.D. Hager and P.N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(10):1025–1039, Oct 1998.

[20] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[21] Thomas Hofmann. Probabilistic latent semantic analysis. In *In Proc. of Uncertainty in Artificial Intelligence, UAI99*, pages 289–296, 1999.

[22] Alex Holub and Gregory Griffin. Caltech 101. `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`, 2006.

[23] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Shape recovery*, pages 389–407, 1992.

[24] Allan D. Jepson, David J. Fleet, and Thomas F. El-Maraghi. Robust online appearance models for visual tracking. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:415, 2001.

[25] F. Jurie and M. Dhome. A simple and efficient template matching algorithm. In *Proc. ICCV*, pages 200–1, 2001.

[26] Frédéric Jurie and Michel Dhome. Hyperplane approximation for template matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):996–1000, 2002.

[27] Yan Ke and R. Sukthankar. Pca-sift: a more distinctive representation for local image descriptors. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 506–513, 2004.

[28] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition*, volume 2, pages 2169–2178, 2006.

[29] Vincent Lepetit and Pascal Fua. Monocular model-based 3d tracking of rigid objects: A survey. In *Foundations and Trends in Computer Graphics and Vision*, pages 1–89, 2005.

[30] David Lowe. SIFT demo program version 4. `http://www.cs.ubc.ca/~lowe/keypoints/`, 2009.

[31] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.

[32] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (darpa). In *Proceedings of the 1981 DARPA Image Understanding Workshop*, pages 121–130, April 1981.

[33] R. Madison, P. DeBitetto, A. Rocco Olean, and M. Peebles. Target geolocation from a small unmanned aircraft system. In *Aerospace Conference, 2008 IEEE*, pages 1–19, March 2008.

[34] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (27):2005.

[35] S. Ravela, B. Draper, J. Lim, and R. Weiss. Adaptive tracking and model registration across distinct aspects. In *International Conference on Intelligent Robots and Systems*, pages 174–180, 1995.

[36] Cordelia Schmid and Roger Mohr. Local greyvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:530–535, 1997.

[37] J. Shi and C. Tomasi. Good features to track. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 593–600, June 1994.

[38] G. Simon, A. W. Fitzgibbon, and A. Zisserman. Markerless tracking using planar structures in the scene. In *Augmented Reality, 2000. (ISAR 2000). Proceedings. IEEE and ACM International Symposium on*, pages 120–128, 2000.

[39] J. Sivic and A. Zisserman. Video Google: Efficient visual search of videos. In J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, editors, *Toward Category-Level Object Recognition*, volume 4170 of *LNCS*, pages 127–144. Springer, 2006.

[40] Richard Szeliski. Computer Vision: Algorithms And Applications. 2009.

[41] P. H. S. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78:2000, 2000.

[42] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large database for non-parametric object and scene recognition. *IEEE PAMI*, 30(11):1958–1970, November 2008.

[43] Michihiro Uenohara and Takeo Kanade. Vision-based object registration for real-time image overlay. In *Computers in Biology and Medicine*, pages 13–22, 1995.

[44] Luca Vacchetti and Vincent Lepetit. Stable real-time 3d tracking using online and offline information. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1385–1391, 2004. Member-Fua, Pascal.

[45] Andrea Vedaldi. Bag of features classification results caltech-101. `http://www.vlfeat.org/~vedaldi/code/bag/cal101.html`, 2009.

[46] P Viola and M Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition*, pages 511–518, 2001.

[47] Changchang Wu. SIFT GPU. `http://www.cs.unc.edu/~ccwu/siftgpu/`, 2009.

[48] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4):13, 2006.

[49] Z Zhang, R Deriche, O Faugeras, and Q-T Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. In *Sensor Signal and Information Processing and a member of the Australian Research Council's Information Technology and Electrical Engineering Large Grants Panel.*, pages 87–119, 1995.

[50] Marco Zuliani. RANSAC toolbox. `http://www.mathworks.com/matlabcentral/fileexchange/18555`, 2009.