USER-TRAINABLE OBJECT RECOGNITION SYSTEMS
VIA GROUP INDUCTION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Alex Teichman
June 2014
I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Sebastian Thrun, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Andrew Ng

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Silvio Savarese

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.
Preface

Despite initial optimism [33] in the 1960s and the five decades of research since, it remains extremely challenging to program computers to perceive and understand objects in the environment around them. While the vast majority of research in this area has been conducted using traditional cameras, high quality depth sensors have recently become available, raising the question of whether they can fundamentally change our approaches to problems in computer vision. The general utility of depth sensors was dramatically demonstrated at the 2005 and 2007 DARPA autonomous driving challenges, in which they were heavily relied upon by all finishing teams for both events. However, their use was primarily limited to low-level collision avoidance. While depth sensors helped provide the position and velocity of obstacles, there was little or no semantic understanding of object type, and as a result the 30,000 pound, six-wheeled autonomous military truck TerraMax spent time meticulously avoiding small tumbleweeds in the road. This was entirely typical.

It was in this context that I began my PhD, investigating how these new sensors could improve the state of the art in object recognition. By happy coincidence, not only was high quality depth sensing now available, but the environment cooperated in a crucial way. Specifically, because objects on roads actively work to avoid collision, it is possible to segment and track most objects seen in depth data without using learned models of object appearance or motion. The availability of this model-free segmentation and tracking enables a new approach to the object recognition problem at the same level of abstraction as the well-known sliding window or semantic segmentation methods. This new approach, named STAC (“segment, track, and classify”), is detailed in Chapter 1. The STAC approach assumes the existence of a frame
classifier which takes as input a single view of an object and produces probabilistic
output regarding semantic label. Track classifications are produced by accumulating
individual frame classifications in an augmented discrete Bayes filter. These ideas are
rigorously evaluated using a new publicly-available dataset, the Stanford Track Col-
lection, for evaluating track classification of cars, pedestrians, and bicyclists versus a
large sampling of everything else an autonomous car sees on typical public roads.

The STAC approach requires some form of model-free segmentation and tracking,
but in return provides three significant advantages. First, annotation of training data
is more efficient because entire tracks rather than individual views can be labeled
with a single keypress. Second, track classification is substantially more accurate
than individual frame classification because of the ability to accumulate predictions
over many different viewpoints, object poses, and occlusion levels. Third, STAC
opens the door to a new and highly effective form of semi-supervised learning, group
induction.

The work of Chapter 1 describes STAC in the fully-supervised regime. About
700,000 individual views of objects were used to train the classifier, requiring approx-
imately 7,000 annotated tracks. While this number is small in some respects, it is an
unacceptably high burden to impose on individuals who might want to train such a
system themselves. However, because of our use of the STAC approach, our unlabeled
data has group structure: it contains groups of unlabeled instances which share the
same hidden semantic label. For example, a track of a bicyclist will remain a bicyclist
throughout, even as viewpoint changes, its rider moves, and occlusions come and go.
It is this structure in unlabeled data that enables group induction. Intuitively, if a
machine learning classifier can confidently identify an entire track as a bicyclist, then
any individual frames within that track that the frame classifier misclassified can be
harvested as new and useful training examples. This intuition is encoded in an alter-
nating optimization problem similar to the expectation-maximization algorithm, but
with the addition of group structure in the unlabeled data. By leveraging large quan-
tities of unlabeled data, group induction can produce accurate classifiers using very
modest amounts of user annotation – typically on the order of one hundred labeled
tracks. This is in range of what end users, rather than corporations or computer
science graduate students, might be willing to provide.

The primary limitation of STAC and group induction is the dependence on model-free segmentation and tracking. While simple segmentation and tracking methods are mostly sufficient in autonomous driving, there are still important failure cases such as crowds of people or cars parked very close to trees. More broadly, most environments beyond autonomous driving do not admit the kind of simple segmentation and tracking methods used in Chapters 1 or 2. Instead, objects of interest are typically found in non-trivial contact with the environment: people with chairs, cups and keyboards with clutter on desks, and so on.

This brings us to the final contribution of this dissertation. How can one get the model-free segmentation and tracking necessary to apply STAC and group induction in these more challenging environments? The STRGEN algorithm (pronounced “sturgeon”) of Chapter 3 provides a method of propagating object segmentations in such environments. STRGEN, short for segmentation and tracking of generic objects, is named such because of the assumption that we have no prior object model to make use of, reflecting our desire to use it in tandem with group induction to learn new object models. To further maximize applicability, we will assume the sensor can have unknown motion and that target objects need not be rigid.

A problem of this difficulty likely requires the use of a wide range of cues: color, texture, 3D shape, image edges, depth edges, surface normal changes, optical flow vectors, and so on can all contribute probabilistically to the propagation of a segmentation mask through time. STRGEN synthesizes a simple and elegant method of combining these diverse cues at runtime as well as – crucially – a method of learning from data how to best combine these cues for use in propagating a segmentation for a randomly selected, as-yet-unseen object. This learning is one level of abstraction higher than the online learning of object models often seen in state of the art bounding box trackers, and is essential to the effective use of the rich segmentation models we consider.

For the sake of telling a single research story without tangents, the content of this dissertation derives almost exclusively from [54], [50, 53], and [56]. As one would expect, the actual research proceeded somewhat less linearly. The first iteration of
what became group induction was designed for only the offline case and had not yet found its rigorous mathematical underpinning [49, 51]. Further, a significant tangent in unsupervised calibration of RGBD sensors can be found in [57, 30]; in many ways these works share similar design goals and intuition to that employed in group induction, and have potential to be significantly useful to practitioners in the field, but they will not be discussed further here.
Acknowledgements

It was my good fortune and great pleasure to arrive in Sebastian Thrun’s lab shortly after the DARPA Urban Challenge. Sebastian is an excellent source of strategic advice and his bold leadership in everything from autonomous driving to online education is inspiring. Junior, the lab’s autonomous car, was an ideal perception research platform at a time when depth sensors were scarce and expensive, providing a literal peek into the future of machine perception. This dissertation would not have been possible without the hard work of the entire autonomous driving team; of particular importance to my work were Jesse Levinson, Mike Montemerlo, Brice Rebsamen, Mike Sokolsky, and Christian Plagemann.

There are many others to whom I owe thanks. Stephen Miller, Jake Lussier, and Jesse Levinson were patient and hard-working co-authors. Neal Parikh provided significant helpful insight along the way. Andrew Ng, Silvio Savarese, Stephen Boyd, and Jerome Friedman rounded out a stellar defense committee. My fellow PhD student interns at Willow Garage in 2008 and 2009 taught me how to get things done in C++, and the entire Willow network made for ever-present conference friends. The robotics middleware framework ROS saved countless hours of my time. My parents, Richard and Jean Teichman, prioritized their children over all else and encouraged my interest in science and technology from a young age. Too many friends to name made exploring the San Francisco Bay Area and surrounding wilderness exceptionally fun.

Finally, Cindy Chung, your companionship over the years has been invaluable. We’ve had so many great adventures together; I look forward to many more!
# Contents

<table>
<thead>
<tr>
<th>Preface</th>
<th>iv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>viii</td>
</tr>
</tbody>
</table>

1 STAC: Segment, Track, and Classify 1

1.1 Introduction ............................. 2
1.1.1 Related Work .......................... 3
1.2 The Stanford Track Collection .............. 5
1.2.1 Details ............................... 6
1.2.2 Track Extraction Process ................ 8
1.3 Classification Methods ..................... 9
1.3.1 Boosting Framework ..................... 9
1.3.2 The Augmented Discrete Bayes Filter ...... 12
1.3.3 Implementation Details .................. 16
1.4 Descriptors .............................. 16
1.4.1 Holistic Descriptors .................... 17
1.4.2 Segment Descriptors .................... 17
1.5 Results .................................. 18
1.6 Conclusions .............................. 22

2 Group Induction 23

2.1 Introduction .............................. 24
2.2 Related work .............................. 25
2.3 Group induction ............................ 26
List of Tables

1.1 Breakdown of the Stanford Track Collection by class. Tracks were collected from busy streets and intersections. ........................................ 7
1.2 Summary of accuracy results for each 1-vs-all problem and for the overall problem. ................................................................. 19
List of Figures

1.1 Example scan of two other cars and a bicyclist at an intersection. Points are colored by return intensity. .................................................. 3
1.2 Junior, the autonomous vehicle research platform. In this work, we use a Velodyne dense LIDAR sensor and an Applanix inertial measurement unit to track objects while moving. .................................................. 5
1.3 Example segments from tracks of objects. Columns contain the three foreground object classes (car, pedestrian, bicyclist) and background objects (everything else). All objects in the environment are tracked, and the task is to classify tracks with the appropriated label. Points are colored by laser return intensity. .................................................. 6
1.4 Histogram of distance to tracked objects in the test set. .................. 7
1.5 The graphical model encoding the independencies in the discrete Bayes filter for our task. Each track has one label $y$, one set of holistic descriptors $w$, and one set of segment descriptors $z_t$ for each segment in the track. $T$ is the total number of segments in the track. ........... 13
1.6 Schematic drawing of the classification process. Log odds estimates from the segment classifier and from the holistic classifier are combined using the augmented discrete Bayes filter. In this example, a distant bicyclist is initially difficult to distinguish from a pedestrian but becomes recognizable as it draws nearer. The holistic classifier, evaluating statistics of the track such as mean speed and maximum acceleration, is fairly confident that the object is not a pedestrian, but relatively uncertain otherwise. A transient occlusion causes an incorrect segment classification, but the overall prediction is correct.

1.7 Examples of side, top, and front views of objects using virtual orthographic camera intensity images. Pixel intensities reflect the average LIDAR return intensity. Several HOG descriptors are computed on each view. Alignment into a canonical orientation enables seeing consistent views of objects.

1.8 Precision-recall curve for classification using the augmented discrete Bayes filter.

1.9 Confusion matrix for classification using the augmented discrete Bayes filter. Entries in the matrix are numbers of tracks.

1.10 Accuracy vs number of points for segment-wise classification. Objects with few points account for the majority of misclassifications.

2.1 We evaluate group induction on an object recognition task using Stanford’s autonomous vehicle, Junior.
2.2 Example of the induction phase for two hypothetical tracks, each of which defines a group of instances assumed to share the same label. Log odds predictions $H(x)$ for each frame were generated using a classifier trained with our method. The right column shows the objective function values corresponding to the optimization problem (2.2). In the top row, $y = +1$ minimizes the objective function so the track is inducted as a pedestrian. The result is that a useful training example - one for which the classifier predicts incorrectly - is added to the training set. In the bottom row, a transient under-segmentation looks very unlike a pedestrian to the classifier, preventing induction; $y = 0$ minimizes the objective function and so the group does not contribute during the training phase.

2.3 Example objects from the Stanford Track Collection [55]. This dataset contains tracks of objects segmented using a standard depth-segmentation algorithm used in autonomous vehicles [28]. Each track forms a group of instances, all of which are assumed to share the same hidden label.

2.4 Intuition for the grid feature transform we use. The bar plot shows histograms of positive and negative examples of cars vs descriptor element value. A naive application of the standard online learning technique of logistic regression assumes a linear relationship between each individual descriptor element and the log odds. That would mean assuming, for example, that as the height of an object increases, the log odds of being a car increases or decreases linearly; this is clearly wrong for our data. By instead having a binary variable per bin, we can learn that cars tend to live within a certain height range. The parametric boosting classifier of Section 2.4.2 builds on this feature transform by using weak classifiers that make predictions for a single bin.
2.5 Supervised accuracy of the parametric boosting classifier vs amount of
training data provided. (Group induction is not used here.) Each line
shows results for a different random ordering of the training data. Para-
metric boosting is similar to that used in previous work [51], but has
the advantage of constant runtime - an essential property for lifelong
learning. This experiment demonstrates that the new formulation can
produce accuracy similar to the old method while delivering an asym-
ptotic speed improvement. It also informs us of the number of training
examples needed to achieve a given level of accuracy with supervised
learning.

2.6 Experimental results showing ability of group induction to adapt to
user feedback and make use of large amounts of unlabeled data. Bar
plots show user annotations, with positive examples in green and neg-
ative examples in gray. Notice, for example, the jump in recall when
new positive examples of pedestrians are added. The retrospection
strategy discussed in Section 2.3.2 is particularly evident in the final
addition of negative examples to the car problem: Very confident false
positive inductions were identified, leading to de-induction of most of
the buffer. Performance then converges to a stable and accurate solu-
tion. Bicyclist and pedestrian plots are clipped to show detail in the
eyearly stages; they remain stable through the end. The final accuracy of
group induction is 98.1%. In contrast, training on just the annotated
examples from this run (i.e. without applying group induction) pro-
duces final accuracy of 88.6%. Predicting $y = -1$ for all class problems
produces final accuracy of 81.8%. Amounts of user-annotated data in
this figure can be fairly compared with those in the fully-supervised
experiment of Figure 2.5.
3.1 Definition of the model-free segmentation and tracking task addressed in this paper. Given an initial seed labeling (first frame; cat on a coffee table), the goal is to produce a segmentation mask of the object over time (subsequent frames) without a pre-trained object model. Foreground points are shown bold and colored, background small and gray.

3.2 Example usage of model-free segmentation and tracking algorithm for object detector training set collection. Initially, no object is being tracked. A user provides sparse foreground and background hints as the RGBD data streams in. Segmented objects are shown in red.

3.3 Visualizations of selected edge and node potentials in the CRF. Edge potentials are zoomed in to show fine structure. Strong edge potentials are darker, while weaker edge potentials fade to original image color. Node potentials expressing a preference for foreground are shown in red, for background in green, and for neither in black. Top row: original image, canny edge potentials, color distance, depth edge potentials. Bottom row: ICP, frame alignment bilateral filter, optical flow, and patch classifier node potentials. Best viewed in color.

3.4 While image appearance and depth information may sometimes provide little information about an object boundary, a change in surface normals can be informative. Surface normals are shown as small dark lines, seen from two angles to aid in depth perception.

3.5 Comparison of our method with the most similar state-of-the-art work. Hamming loss is absolute number of pixels wrong; normalized accuracy takes into account object size and has a maximum of one. Total error reduction is about 65%.

3.6 Comparison of our method with the most similar state-of-the-art work when dropping node and edge potentials that make use of depth data. Total error reduction is about 40%.

3.7 Visualization of results in which image appearance alone would lead to a very difficult segmentation problem, but which becomes relatively easy when reasoning about depth as well.
3.8 Visualization of results. The first frame in each sequence (far left) is the seed frame. Foreground points are shown in bold and color while background points are shown small and gray. Best viewed on-screen.

3.9 Visualization of the original method versus the real-time version. Node potential and edge potential visualizations follow the same conventions as in Figure 3.3. Best viewed in color.

3.10 Test set normalized loss and average per-frame segmentation time compared with different levels of algorithmic optimization as discussed in Section 3.4.2. Thirteen runs of each condition were made; error bars and annotations show one standard deviation for this dataset. While a 75% downsampling induces too much error to be acceptable, 50% downsampling appears to be a good trade-off for many applications. Timing results for the original method differ from those presented in Section 3.4.1 because multithreading is enabled here.

3.11 Average frame segmentation timing breakdown for VGA and QQVGA resolutions using the boundary mask and 50% downsampling of node potentials. Feature computation dominates and the graph cuts solver does not preclude real-time operation even at VGA resolution; thus, real-time segmentation at higher resolutions is conceivable with further optimizations to the feature set. Best viewed in color.

3.12 Example data labeling pipeline demonstrating one possible application of the STRGEN algorithm. Segmented objects are obtained while the raw data is being recorded. Segmentation hints from the user generate node potentials in the CRF so the user can direct the segmentation at any time; without input, the segmenter makes use of only the other cues discussed in Section 3.3.3 to produce segmentation masks. The output is a set of segmented object examples suitable for training standard object detectors.
3.13 Object detection example trained using the method in Figure 3.12. Color and depth templates are extracted from segmented training objects; an appropriate object detection method (LINE-MOD in our example) searches for the templates in previously-unseen test frames and identifies “detections” as matches with responses exceeding a threshold.
Chapter 1

STAC: Segment, Track, and Classify

Abstract

Object recognition is a critical next step for autonomous robots, but a solution to the problem has remained elusive. Prior 3D-sensor-based work largely classifies individual point cloud segments or uses class-specific trackers. In this paper, we take the approach of classifying the tracks of all visible objects. Our new track classification method, based on a mathematically principled method of combining log odds estimators, is fast enough for real time use, is non-specific to object class, and performs well (98.5% accuracy) on the task of classifying correctly-tracked, well-segmented objects into car, pedestrian, bicyclist, and background classes.

We evaluate the classifier’s performance using the Stanford Track Collection, a new dataset of about 1.3 million labeled point clouds in about 14,000 tracks recorded from an autonomous vehicle research platform. This dataset, which we make publicly available, contains tracks extracted from about one hour of 360-degree, 10Hz depth information recorded both while driving on busy campus streets and parked at busy intersections.
CHAPTER 1. STAC: SEGMENT, TRACK, AND CLASSIFY

1.1 Introduction

Object recognition in dynamic environments is one of the primary unsolved challenges facing the robotics community. A solution would include segmentation, tracking, and classification components, and would allow for the addition of new object classes without the need for an expert to specify new models. Joint solutions are currently impractical; however, if the classification of well-segmented, correctly-tracked objects were solved, there is a clear path towards a joint solution using EM-like methods or multi-hypothesis trackers.

The recent introduction of dense 3D sensors makes now an ideal time to explore a large dataset approach to track classification. These sensors enable data-driven segmentation and tracking of arbitrary objects – that is, all objects in the environment regardless of object class. This is especially effective in the driving context, where objects often actively work to stay well-segmented from each other. Tracking and segmentation greatly facilitate the process of labeling large amounts of data, as labeling an object which has been correctly segmented and tracked can provide hundreds or thousands of training examples in a single keypress.

Our particular task of track classification is inspired by the autonomous driving problem, in which it is useful to track all objects in the environment and additionally recognize some object classes that must be treated specially. We address a multiclass problem in which tracks must be classified as car, pedestrian, bicyclist, or background (any other object class).

None of the algorithms described here are specifically adapted to any particular dense 3D sensor, and the addition of new object classes requires only a labeled dataset of tracks that include the new object class. In this sense, our algorithms apply well beyond our particular sensors and classification task.

The contributions of this paper are three-fold. First, we describe a novel approach to the classification of arbitrary object tracks which performs well on a large, real-world data set. This algorithm makes progress towards the overall goal of object recognition in dynamic 3D environments. Second, we adapt the ideas of hybrid
CHAPTER 1. STAC: SEGMENT, TRACK, AND CLASSIFY

3

Figure 1.1: Example scan of two other cars and a bicyclist at an intersection. Points are colored by return intensity.

generative / discriminative models [36] to alleviate the overly strong conditional independence assumptions made when combining log odds estimates over time. Third, we provide a large dataset of labeled and tracked 3D point clouds to the research community.

1.1.1 Related Work

Current object recognition methods generally fall into two categories. First, multi-class object recognition in static scenes continues to be an active area of research, e.g. in [26, 59, 14, 44, 15]. One path to object recognition in dynamic scenes is to solve static scene object recognition so well that tracking detections is sufficient, as in [4, 17]. However, this approach is currently not possible for general object recognition,
and the temporal nature of our task offers much useful information.

Second, object recognition via the tracking of specific object classes has been well-studied, e.g. in [2, 34, 64, 41, 40], but these methods require the explicit specification of new tracker models for each additional object class. In general, object recognition in dynamic scenes using class-specific trackers will (presumably) obtain better performance for the classes they are designed for, but at the cost of additional expert time to specify new models if they are needed. For specific systems, e.g. those that are only concerned with recognizing humans for a human-robot interaction task, this is an entirely appropriate tradeoff; however, in this paper, we propose a classification method that can increase the number of classes without necessitating onerous work by an object recognition expert.

There is only limited prior work on methods that classify tracks of arbitrary objects. The work of [29] considers the task of learning exemplar models for arbitrary object tracks in an unsupervised way. In contrast, the method in this paper is fully supervised, but can handle the large numbers of diverse distractor objects that are seen in operation of the autonomous vehicle on typical streets. The most similar work to ours, [45], fuses 2D laser and camera imagery to track pedestrians and cars at ranges less than 15 meters. It is clear that there remains a large gap between the state of the art and what would be required for a practical, high-reliability object recognition implementation that works at long range. Our work helps close this gap by demonstrating high track classification performance on a very large dataset set at longer ranges than have previously been considered.

This paper is structured as follows. In Section 1.2, we discuss the Stanford Track Collection and the track extraction process. In Section 1.3 we describe the log odds estimator (boosting) and our method of combining boosting outputs. In Section 1.4 we review the descriptors used by the boosting classifiers. Finally, results and conclusions are presented in Sections 1.5 and 1.6.
Figure 1.2: Junior, the autonomous vehicle research platform. In this work, we use a Velodyne dense LIDAR sensor and an Applanix inertial measurement unit to track objects while moving.

1.2 The Stanford Track Collection

The purpose of this dataset is to provide a large-scale testing ground for track classification in busy street scenes. The dataset includes over 1.3 million labeled segmented objects across roughly 14,000 tracks, with a maximum range of about 70 meters and mean range of about 30 meters. Table 1.1 details the breakdown by class, and Figure 1.4 shows the distribution of range to objects. The large percentage of tracks that do not fall into the car, pedestrian, or bicyclist classes (83%) is a natural consequence of collecting the dataset directly from an autonomous vehicle research platform while driving on normal streets; the class distribution thus reflects that of a vehicle’s typical environment. The dataset is available at [55].

In the following sections, we will make use of several terms: a scan is the set of depth measurements recorded during one 360-degree revolution of the dense 3D sensor; a track is a temporal series of 3D point clouds of the same object, derived from segmentation of the scans; and a segment is a single point cloud from a track.
1.2.1 Details

To ensure valid results, the training set and test set are drawn from different locations at different times. Pooling all tracks, then drawing them at random into the test and training sets would be flawed; it is common for several distinct tracks to exist for the same object, as occlusions can cause a tracked object to be lost and then picked up as a new track.

The details of the labeling in this dataset are motivated by the autonomous driving task. For example, because it is relatively common for two pedestrians to be segmented together, this occurrence is labeled as a pedestrian in the dataset. Similarly, rollerbladers, skateboarders, pedestrians walking bicycles, pedestrians pulling suitcases, and pedestrians on crutches are all labeled as pedestrians in this dataset.
### Number of tracks

<table>
<thead>
<tr>
<th>Set</th>
<th>Car</th>
<th>Pedestrian</th>
<th>Bicyclist</th>
<th>Background</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>855</td>
<td>202</td>
<td>177</td>
<td>6146</td>
<td>7380</td>
</tr>
<tr>
<td>Test</td>
<td>823</td>
<td>120</td>
<td>141</td>
<td>4984</td>
<td>6068</td>
</tr>
<tr>
<td>Total</td>
<td>1678</td>
<td>322</td>
<td>318</td>
<td>11130</td>
<td>13448</td>
</tr>
</tbody>
</table>

### Number of frames

<table>
<thead>
<tr>
<th>Set</th>
<th>Car</th>
<th>Pedestrian</th>
<th>Bicyclist</th>
<th>Background</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>89008</td>
<td>32107</td>
<td>30496</td>
<td>521624</td>
<td>673235</td>
</tr>
<tr>
<td>Test</td>
<td>57480</td>
<td>23102</td>
<td>25553</td>
<td>528693</td>
<td>634828</td>
</tr>
<tr>
<td>Total</td>
<td>146488</td>
<td>55209</td>
<td>56049</td>
<td>1050317</td>
<td>1308063</td>
</tr>
</tbody>
</table>

Table 1.1: Breakdown of the Stanford Track Collection by class. Tracks were collected from busy streets and intersections.

![Figure 1.4: Histogram of distance to tracked objects in the test set.](image-url)
The car class contains sedans, vans, trucks, cars with trailers, etc., but does not include tractor trailers or large buses. There are many reasonable choices of where to draw class lines, and do not believe any particular choice is essential to this work provided it is consistently applied.

To avoid confounding failures of the track classification method with failures of the tracking and segmentation components of a full object recognition system, we invalidate foreground tracks (those of cars, pedestrians, and bicyclists) in which more than 10% of the segments are incorrect due to tracking or under-segmentation errors. The over-segmentation of an object, i.e. when an object is segmented into more than one part, is not considered a segmentation error, as this is a common case that must be dealt with in practice. Additionally, with background tracks, what constitutes a tracking or segmentation error is frequently ambiguous. As this case is also a common one that a track classifier must handle correctly, tracking and segmentation errors of background objects are not considered inconsistent. About 350 inconsistent tracks exist in the dataset; this constitutes about 2% of the entire dataset or about 13% of foreground tracks.

Finally, we require that all tracks have at least ten segments (i.e. the objects is seen for at least one second) and that at least one of these segments has greater than 75 points.

1.2.2 Track Extraction Process

Our research vehicle, shown in Figure 1.2, senses the environment with a Velodyne HDL-64E S2, a dense 64-beam scanning LIDAR that provides 360-degree coverage at 10 Hz, generating just over 1 million 3D points with associated infrared remittance values per second. A tightly coupled GPS/IMU pose system, the Applanix LV-420, provides inertial updates and global position estimates at 200 Hz, enabling tracking while the vehicle is moving. The LIDAR is calibrated using the algorithm of [27].

Scans are segmented using a connected components algorithm on an obstacle map. The 2D obstacle grid is built using standard efficient methods, then clustered with a flood fill algorithm. Clusters that are too large or too small to be of interest are
CHAPTER 1. STAC: SEGMENT, TRACK, AND CLASSIFY

Tracking of cluster centroids is accomplished using linear Kalman filters. The data association problem is resolved by distance from measurement centroid to the expected position of each filter. Any cluster not matched to a filter spawns a new filter. Any filter not matched to a cluster has its prediction step run but not its update step, increasing the uncertainty in position; the filter is removed after position uncertainty exceeds a threshold.

We note that the specific segmentation and tracking algorithms discussed here are not central to our overall approach, and more sophisticated algorithms could be readily substituted in different contexts.

1.3 Classification Methods

We first describe the boosting methods used to classify individual segments, and then we describe how the outputs of the boosting classifiers are combined into a final log odds estimate for a track. We make some references to the descriptors used in these sections, but defer those details to Section 1.4.

1.3.1 Boosting Framework

Boosting is a method of combining the predictions of many weak classifiers into a single, higher-accuracy strong classifier. Intuitively, the weak classifiers encode simple rules: for example, “if the object has a large bounding box, then it is probably not a pedestrian,” or “an object that has a side profile that looks like this one is probably a bicyclist.”

We use two boosting classifiers in our system. First, the segment classifier makes a prediction when given a set of segment descriptors, i.e. a set of vectors which describe the appearance of an object at a point in time. Second, the holistic classifier makes a prediction when given a set of holistic descriptors of a track, i.e. a set of vectors that describe speed, acceleration, and other properties of the track as a whole.

Taking the optimization perspective on boosting, in a two-class problem we desire
a solution to

$$\min_H \mathbb{E} \left[ \exp \left( -\frac{1}{2} Y H(Z) \right) \right], \tag{1.1}$$

where the expectation is over the joint distribution, $Y$ is the class label, $Z$ is a set of
descriptors, and $H$ is the strong classifier. The $1/2$ term causes the optimal solution to
this problem to be $H(z) = \log \frac{P(Y=1|z)}{P(Y=-1|z)}$; see [16] for details. In this paper, we assume
all boosting classifiers use this form and thus output log odds estimates rather than
the usual $1/2$ log odds.

In practice, we approximate the expectation with a sum, resulting in

$$\min_H \sum_{m=1}^{M} \exp \left( -\frac{1}{2} y_m H(z_m) \right),$$

with training examples $(y_m, z_m)$, where $y_m \in \{-1, +1\}$ is the class label and $z_m$ is the
descriptor set for training example $m$.

Extending to the multiclass boosting formulation, we use a 1-vs-all scheme, resulting in
the optimization problem

$$\min_H \sum_{c=1}^{C} \sum_{m=1}^{M} \exp \left( -\frac{1}{2} y_m^c H(z_m, c) \right).$$

Here, the strong classifier $H$ makes a prediction for each class $c$. The integer class
label $y_m$ is one of $\{1, 2, \ldots, C\}$. If $y_m = c$, $y_m^c = +1$; otherwise $y_m^c = -1$. Background
objects have $y_m^c = -1$ for all $c$.

We use a variant of GentleBoost [16] and JointBoost [58] to handle the multi-
class, multi-descriptor, high-dimensional nature of the problem. As in JointBoost,
the weak classifiers are shared across the three 1-vs-all classification problems; un-
like JointBoost, since we do not consider problems with extremely large numbers of
classes, we force all weak classifiers to make predictions for all 1-vs-all classification
problems.

As is usual in boosting, the strong classifier $H$ is defined as a sum of the weak
classifiers $h_k$ for all $k \in \{1, 2, \ldots, K - 1\}$, i.e.

$$H(z, c) = \sum_k h_k(z, c).$$

Our weak classifiers take the form

$$h_k(z, c) = \begin{cases} 
   a^c_k & \text{if } ||f_k(z) - x_k||_2 \leq \theta_k \\
   0 & \text{otherwise} 
\end{cases},$$

where $\theta_k$ is in $\mathbb{R}$, $f_k$ chooses a particular descriptor in the set $z$, and $x_k$ is a descriptor in the same space as $f_k(z)$. The response value $a^c_k$ is positive if the weak classifier predicts the descriptor to be of class $c$, and negative otherwise; $|a^c_k|$ is the confidence of the prediction. Geometrically, this means that a weak classifier makes a prediction $a^c_k$ about a descriptor $f_k(z)$ if the descriptor falls within a hypersphere of radius $\theta_k$ centered at the point $x_k$.

The strong classifier can be imagined as a set of different-sized hyperspheres living across many descriptors spaces of different dimensions. A test example will have a descriptor in each of these descriptor spaces, and the weak classifier hyperspheres which contain it all respond with their own predictions for each class.

The strong classifier is learned using the procedure of GentleBoost [16]; new weak classifiers are iteratively added by solving

$$\min_{f_K, x_K, \theta_K, \vec{a}_K} \sum_c \sum_m w^c_m \exp\left(-\frac{1}{2}y^c_m h_K(z_m, c)\right),$$

where

$$w^c_m = \exp\left(-\frac{1}{2}y^c_m \sum_k h_k(z_m, c)\right)$$

is the multiclass boosting weight.

This problem is solved through a combination of directed search, dynamic programming, and convex optimization. We wish to direct the learning process to focus its attention on difficult examples. Concretely, a training example is sampled from the boosting weights distribution and a descriptor space is chosen uniformly at random; this specifies an $f_K$ and an $x_K$. Given $f_K$, $x_K$, and $\theta_K$, the value of $\vec{a}_K$ can be found analytically with a single Newton step for each class. Dynamic programming
efficiently solves for $\vec{a}_K$ for each possible $\theta$, i.e. one $\theta$ per training example other $x_K$. This process is repeated for a number of choices of $(f_K, x_K)$ pairs. The best evaluated choice of $(f_K, x_K, \theta_K, \vec{a}_K)$ in terms of the optimization objective (1.2) is then added as a new weak classifier.

1.3.2 The Augmented Discrete Bayes Filter

We have now described a method for producing log odds estimates given the descriptors of a segment (e.g. object shape) or the descriptors of a track as a whole (e.g. mean speed). What remains is to specify a principled and effective method of combining these into a final log odds estimate for a track.

Central to our approach is the fact that a boosting classifier outputs an estimate of the log odds. This enables their combination with a discrete Bayes filter. Other log odds estimators, such as logistic regression, could be substituted into this framework.

To simplify the notation, the following section considers only two-class models; the extension to multiclass is straightforward.
Figure 1.5: The graphical model encoding the independencies in the discrete Bayes filter for our task. Each track has one label $y$, one set of holistic descriptors $w$, and one set of segment descriptors $z_t$ for each segment in the track. $T$ is the total number of segments in the track.

For reference, we provide the definitions of several symbols below.

- $z_t$ – The segment descriptors for the $t$-th segment in a track. These include spin images, bounding box size, and virtual orthographic camera views.
- $w$ – The holistic descriptors of a track, which include mean speed, maximum speed, maximum angular velocity, etc.
- $\mathcal{L}(x) = \log \frac{P(Y=1|x)}{P(Y=-1|x)}$ – The log odds given $x$, for any choice of $x$.
- $\mathcal{L}_0 = \log \frac{P(Y=1)}{P(Y=-1)}$ – The log prior odds.
- $H^H$ – The holistic classifier. Returns a log odds estimate given the holistic descriptors $w$ of a track.
- $H^S$ – The segment classifier. Returns a log odds estimate given the segment descriptors $z_t$ of segment $t$.
- $\mathcal{L}_0^S$ – The empirical estimate of the log prior odds for the segment classifier.
- $\mathcal{L}_0^H$ – The empirical estimate of the log prior odds for the holistic classifier.

As in the discrete Bayes filter, the independencies of the model are the same as those in naïve Bayes; see Figure 1.5.
We desire an estimate of $\mathcal{L}(w, z_{1:T})$, the log odds given all the information known about a track. Expanding this term out, we have

$$
\mathcal{L}(w, z_{1:T}) = \log \frac{P(Y = 1|w, z_{1:T})}{P(Y = -1|w, z_{1:T})}
= \mathcal{L}_0 + \log \frac{P(w, z_{1:T}|Y = 1)}{P(w, z_{1:T}|Y = -1)}
= \mathcal{L}_0 + \log \frac{P(w|Y = 1)}{P(w|Y = -1)} + \sum_{t=1}^{T} \log \frac{P(z_t|Y = 1)}{P(z_t|Y = -1)}
= \mathcal{L}(w) + \sum_{t=1}^{T} (\mathcal{L}(z_t) - \mathcal{L}_0)
\approx H^H(w) + \sum_{t=1}^{T} \left( H^S(z_t) - \mathcal{L}_0^S \right). \tag{1.3}
$$

The naïve discrete Bayes filter, (1.3), is arrived at from $\mathcal{L}(w, z_{1:T})$ via Bayes’ rule, the conditional independence assumptions shown in Figure 1.5, and then another application of Bayes’ rule. The final approximation is a result of substituting in the holistic classifier $H^H$ and the segment classifier $H^S$, which provide estimates of the log odds given the holistic descriptors and the segment descriptors, respectively.

A readily apparent shortcoming of the naïve discrete Bayes filter is that the holistic classifier $H^H$ becomes increasingly negligible as track length increases. If the conditional independence assumptions were strictly correct, this behavior would be desired. However, due to a variety of possible dependencies between segments that arise in practice, the individual segment classifier estimates are not truly conditionally independent given the object class. This shortcoming is addressed in the normalized discrete Bayes filter,

$$
H^H(w) + \frac{1}{T} \sum_{t=1}^{T} \left( H^S(z_t) - \mathcal{L}_0^S \right).
$$

Finally, we notice that the (again, incorrect) conditional independence assumption $W \perp Z_{1:T} | Y$ remains, and that we can compensate for this approximation by learning additional parameters $\alpha, \beta, \gamma \in \mathbb{R}$ that weight the relative importance of
each term. This gives rise to the augmented discrete Bayes filter (ADBF),

$$H^A(w, z_{1:T}) = \alpha L_0^H + \beta (H^H(w) - L_0^H)$$

$$+ \gamma \frac{1}{T} \sum_{t=1}^{T} (H^S(z_t) - L_0^S). \tag{1.4}$$

The weights are learned via the unconstrained convex optimization problem

$$\min_{\alpha, \beta, \gamma} \sum_m \log \left(1 + \exp \left(-y_m H^A(w_m, z^m_{1:T})\right)\right). \tag{1.5}$$

This formulation is equivalent to logistic regression on the output of the holistic and segment classifiers with an intercept term. We note that the optimization problem

$$\min_{H} \mathbb{E} \left[ \log \left(1 + e^{-YH(Z)}\right) \right]$$

has the same optimal solution of $H(z) = \log \frac{p(Y=1|z)}{p(Y=-1|z)}$ as in (1.1); therefore the final augmented discrete Bayes filter is optimized to produce the best log odds estimate possible.

In summary, to obtain a log odds estimator for a track, we cannot simply throw all descriptors into a logistic regression or boosting model because the length of a track is variable. The discrete Bayes filter handles the variable length of tracks nicely, but, like naïve Bayes, makes strong conditional independence assumptions that are frequently incorrect. Like logistic regression, the ADBF learns parameters that can compensate for these mistakes to some extent.

The ADBF builds on ideas used in [36] for document classification, in which weights are learned to combine terms in a multinomial naïve Bayes model for different regions of a document. Here, we use a comparatively complex discriminative model in place of the simple generative model of multinomial naïve Bayes, and combine entirely different models that operate on different descriptor sets.
Figure 1.6: Schematic drawing of the classification process. Log odds estimates from the segment classifier and from the holistic classifier are combined using the augmented discrete Bayes filter. In this example, a distant bicyclist is initially difficult to distinguish from a pedestrian but becomes recognizable as it draws nearer. The holistic classifier, evaluating statistics of the track such as mean speed and maximum acceleration, is fairly confident that the object is not a pedestrian, but relatively uncertain otherwise. A transient occlusion causes an incorrect segment classification, but the overall prediction is correct.

1.3.3 Implementation Details

The number of candidate weak classifier centers – that is, the number of \((f_K, x_K)\) pairs – evaluated at each round of boosting was set to 20. The segment classifier was trained until 1000 weak classifiers, and the holistic classifier until 100. To learn the weights in the ADBF, part of the training set was used for training segment and holistic classifiers and the other part was used in the optimization problem (1.5).

We found that, in training the segment classifier, using a random sampling of 20% of the training segments did not hurt performance, likely because tracks were sampled at about 10Hz and the variation from one segment to the next is relatively low.

Training of the segment classifier is the slowest part of the training procedure and takes on the order of a few hours in a multithreaded implementation running on an octocore desktop.

1.4 Descriptors

In this section, we review the descriptors used for the segment and holistic boosting classifiers. In general, we believe that the details of the descriptors are not as important as the availability of a wide range of descriptors for the classifier to consider.
1.4.1 Holistic Descriptors

Holistic descriptors describe the track as a whole. In our system, these include (a) maximum speed, (b) mean speed, (c) maximum acceleration, (d) mean acceleration, (e) maximum angular velocity, and (f) the segment descriptors applied to the accumulated point cloud of all segments in the track. Each motion descriptor is computed relative to a velocity estimate of the track’s centroid smoothed over five consecutive segments, or about half a second. We note that there is significant noise in these estimates due to occlusions and segmentation jitter.

The accumulated segment descriptors, while unintuitive for moving objects, can provide complete views of stationary objects that are badly occluded in most or all individual segments.

1.4.2 Segment Descriptors

We use a total of 29 segment descriptors which encode various aspects of object appearance: oriented bounding box size, 4 different parameterizations of spin images [22], and 24 different parameterizations of the histogram of oriented gradients [11]
descriptor computed on virtual orthographic camera intensity images.

Central to the descriptor pipeline is the concept of a canonical orientation of a point cloud. In a coordinate system where $z$ is up, the canonical orientation is a rotation of the segment around the $z$ axis in which the long and short sides of the object are aligned with the $x$ and $y$ axes. We use a simple RANSAC algorithm to fit the dominant line in the XY-projection of the segment’s points.

Virtual orthographic camera images of the front, side, and top views are taken relative to the canonical orientation of the segment. As such, descriptors derived from these images are invariant to rotations of the 3D object around the vertical axis. Image pixels measure average brightness of the return intensities from the sensor. Examples of the orthographic intensity images are shown in Figure 1.7.

The histogram of oriented gradients descriptor captures local object appearance in images with invariance to small translations in the gradient locations. A window size and offset from the edges of the projected segment are specified to capture different parts of an object, such as the front or back wheel of a bicycle; in some cases, these descriptors will be invariant to partial occlusion.

Spin images are all taken about the vertical axis, as other rotations are generally significant in the driving context; cars rarely appear upside down, for example. Spin images are “whitened”, i.e. scaled to have zero mean and unit variance, so they will be invariant to point cloud density.

1.5 Results

Performance was evaluated on the Stanford Track Collection. An example of the classification process is shown in Figure 1.6. The prediction for each track is taken to be the class with the maximum log odds, or background if the log odds for all classes are negative. We note that the log odds approach presented in Section 1.3.2 is more naturally suited to the case in which a track can have multiple positive classifications. For example, it may be useful to have both “car” and “police car” classes, where instances of the latter are also instances of the former.

Because the dataset is drawn from the tracks recorded in actual driving sequences,
Track classification

<table>
<thead>
<tr>
<th>Method</th>
<th>Car</th>
<th>Pedestrian</th>
<th>Bicyclist</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBF</td>
<td>98.7%</td>
<td>99.8%</td>
<td>99.9%</td>
<td>98.5%</td>
</tr>
<tr>
<td>Normalized DBF</td>
<td>98.0%</td>
<td>99.8%</td>
<td>99.8%</td>
<td>97.7%</td>
</tr>
<tr>
<td>Naïve DBF</td>
<td>98.3%</td>
<td>99.6%</td>
<td>99.8%</td>
<td>97.7%</td>
</tr>
<tr>
<td>Segment classifier only</td>
<td>98.3%</td>
<td>99.5%</td>
<td>99.8%</td>
<td>97.6%</td>
</tr>
<tr>
<td>Holistic classifier only</td>
<td>94.3%</td>
<td>99.2%</td>
<td>99.2%</td>
<td>93.0%</td>
</tr>
<tr>
<td>Prior only</td>
<td>86.0%</td>
<td>98.1%</td>
<td>97.7%</td>
<td>81.8%</td>
</tr>
</tbody>
</table>

Single segment classification

<table>
<thead>
<tr>
<th>Method</th>
<th>Car</th>
<th>Pedestrian</th>
<th>Bicyclist</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBF$^1$</td>
<td>97.9%</td>
<td>99.6%</td>
<td>99.7%</td>
<td>97.5%</td>
</tr>
<tr>
<td>Segment classifier only</td>
<td>95.8%</td>
<td>98.3%</td>
<td>98.4%</td>
<td>93.1%</td>
</tr>
<tr>
<td>Prior only</td>
<td>90.7%</td>
<td>96.5%</td>
<td>96.0%</td>
<td>83.3%</td>
</tr>
</tbody>
</table>

Table 1.2: Summary of accuracy results for each 1-vs-all problem and for the overall problem.

A large number of background tracks populate the test set. As such, a classifier which simply predicts background for all test tracks would be 81.8% accurate.

Accuracy results are summarized in Table 1.2; the ADBF achieves 98.5% accuracy. Precision-recall curves for the three classes and a confusion matrix are shown in Figures 1.8 and 1.9, respectively. The holistic classifier alone reaches only 93.0% accuracy; this method is insufficient on its own, but when combined properly with the segment classifier can improve performance.

Classifying individual segments results in 93.1% accuracy, vs 97.5% accuracy when classifying with the augmented discrete Bayes filter, an error reduction of about 65%.$^2$

Perhaps the largest failure case is illustrated by Figure 1.10, which shows degradation of the segment classifier’s accuracy as the number of observed points in the segment decreases. This result includes both long-range and badly occluded objects, both of which tend to be difficult test cases for the classifier.

$^1$For comparison to the other segment-wise results, the classification for the track is applied to each individual segment.

$^2$While a labeled track is allowed to have up to 10% of its segments be incorrect due to segmentation or tracking errors, the total percentage of incorrectly labeled segments due to this cutoff is below 1%.
CHAPTER 1. STAC: SEGMENT, TRACK, AND CLASSIFY

Recall

Precision

Figure 1.8: Precision-recall curve for classification using the augmented discrete Bayes filter.

Figure 1.9: Confusion matrix for classification using the augmented discrete Bayes filter. Entries in the matrix are numbers of tracks.
CHAPTER 1. STAC: SEGMENT, TRACK, AND CLASSIFY

Figure 1.10: Accuracy vs number of points for segment-wise classification. Objects with few points account for the majority of misclassifications.

Classification speed is fast enough to enable real-time use of the system when considering only objects on the road. The average time to compute a classification for a segment, including descriptor computation and amortizing holistic descriptor computation time across the entire track, is about 2 milliseconds in our multithreaded implementation on an octocore computer. Caching of shared computation in the descriptor pipeline is essential.

There are a number of reasons for the strong performance of our system. Segmentations of dense 3D data make possible descriptors that see objects from consistent viewpoints, as we use here with the virtual orthographic camera intensity images. Segmentation also provides more consistent descriptors since background noise is removed; in contrast, consider the case of HOG descriptors of pedestrians in camera images, which can have widely varying backgrounds (and therefore widely varying descriptors) for the same pedestrian. Classifying tracks rather than individual segments allows multiple views of an objects to contribute to the final prediction and is a key component of achieving high performance.
1.6 Conclusions

We have developed a new multiclass track classification method which demonstrates high accuracy on a large, real-world dataset. Our method, which relies on segmentation and tracking made possible by dense 3D sensors, provides a component of a joint solution to 3D object recognition, which includes segmentation, tracking, and classification. The addition of new object classes does not require the specification of new tracker models, but rather only supplemental labeled tracks of the new object class. Finally, our method is fast enough for real time use in plausible implementations.

In this work, we have assumed one possible decomposition of the full object recognition problem - namely, into segmentation, tracking, and track classification components. Our experiments demonstrate an effective method of track classification given correct segmentation and tracking. Initial work on the full object recognition problem indicates that the majority of classification errors are due to failures of segmentation and tracking, so that direction is one of great interest for making progress on object recognition. Additionally, better occlusion-invariant descriptors would be useful. Dense 3D sensors should enable this work, since depth information can provide occlusion labels.
Chapter 2

Group Induction

Abstract

Machine perception often requires a large amount of user-annotated data which is time-consuming, difficult, or expensive to collect. Perception systems should be easy to train by regular users, and this is currently far from the case.

Our previous work, tracking-based semi-supervised learning [51], helped reduce the labeling burden by using tracking information to harvest new and useful training examples. However, [51] was designed for offline use; it assumed a fixed amount of unlabeled data and did not allow for corrections from users.

In many practical robot perception scenarios we A) desire continuous learning over a long period of time, B) have a stream of unlabeled sensor data available rather than a fixed dataset, and C) are willing to periodically provide a small number of new training examples.

In light of this, we present group induction, a new mathematical framework that rigorously encodes the intuition of [51] in an alternating optimization problem similar to expectation maximization (EM), but with the assumption that the unlabeled data comes in groups of instances that share the same hidden label. The mathematics suggest several improvements to the original heuristic algorithm, and make clear how to handle user interaction and streams of unlabeled data. We evaluate group induction on a track classification task from natural street scenes, demonstrating its ability to learn continuously, adapt to user feedback, and accurately recognize objects of interest.
2.1 Introduction

It is not uncommon for perception systems to be trained on thousands or tens of thousands of labeled examples. At the more extreme end of the spectrum, previous supervised work on car, pedestrian, and bicyclist detection in autonomous driving was trained on 600 million labeled instances of objects [54]. Requiring thousands or more training examples makes scaling up these perception systems to larger numbers of object classes an onerous task, and puts the training of object recognition systems firmly out of reach of regular people.

For example, consider a farming robot designed for automated weeding. Each farm will have a different perception task due to differences in soil color, weed populations, crop types, crop age, and so on. Training a new perception system for each farm from scratch using fully-supervised techniques could easily be economically infeasible due to the costs of collecting large numbers of hand-labeled training examples. Typically, training examples for many different viewpoints, lighting conditions, and poses are necessary to produce a high accuracy perception system. However, a few examples provided by a user should be enough for the robot to recognize at least one object of interest, then look at it from different angles to automatically harvest new and useful training examples. It is this intuition that [51] harnessed, and which we will continue to exploit in this paper. The goal is accurate perception systems that can be trained by non-experts using only tens of training examples. Towards this end, the primary contributions of this work are:

- a more rigorous mathematical framework for the intuition behind tracking-based semi-supervised learning, generic to any scenario in which the unlabeled data has group structure.
- extensions of this math for making use of streams of unlabeled data and occasional user input.
- a particular implementation of the above which we empirically demonstrate is practical for lifelong learning of object recognition systems for autonomous driving.
CHAPTER 2. GROUP INDUCTION

Figure 2.1: We evaluate group induction on an object recognition task using Stanford’s autonomous vehicle, Junior.

2.2 Related work

Group induction makes use of both labeled and unlabeled data and therefore falls into the category of semisupervised methods, for which a broad overview can be found in [66]. The abstract math is perhaps most related to the EM algorithm [12]; both are alternating optimization problems with training and hidden variable estimation steps, but group induction additionally assumes the group structure in the unlabeled data and uses hard rather than soft assignment.

Co-training [7] is a related semisupervised method which uses different “views” of the same instances. A different classifier for each view is maintained and used for labeling the training set of the other view. An example from a robotics context would be having a camera view and a pointcloud view of every training example; then the system could, in theory, be trained with camera images of cars, learn to recognize them in laser range finder data, then harvest camera image training examples of car headlights at night. This is similar to group induction in that groups of (two) unlabeled instances are assumed to have the same label, but different in that the two
unlabeled instances live in different descriptor spaces. In group induction, we assume that all instances live in the same descriptor space.

Active learning involves a machine learning system asking a user for labels to particular examples. A broad overview can be found in [42]. We make use of a very simple version of active learning in which, when a user decides to provide feedback to the system, examples are sorted by confidence; useful examples can easily be found in low-confidence regions of the space. In robotics, active learning is used, for example, in [13] to reduce the labeling burden for autonomous navigation.

Other ways to reduce the labeling burden have also been explored in robotics. For example, in [26], domain adaptation is applied to use 3D models to improve object recognition in laser scans. Self-supervised learning is also common in terrain classification, as terrain labels can be found automatically using depth sensors [63, 10] or accelerometers [65].

A preliminary version of the work in this paper was presented at an RSS 2012 workshop as a presentation only. Slides can be found at [52]. The goals of online and active learning were the same, but group induction had not yet been developed.

### 2.3 Group induction

In this section, we introduce the abstract mathematics of group induction. In Section 2.5.2, we will consider a concrete instantiation of this math for the autonomous driving problem.

Intuitively, group induction is a semi-supervised framework for harvesting new and useful training examples from unlabeled data by exploiting the fact that groups of unlabeled examples are known to share the same (hidden) label. Initially, a classifier is trained using only a small amount of user-annotated data. This initial classifier is used to search the unlabeled dataset for groups it can classify confidently. These groups get added to the training set, the classifier is updated using the new examples, and the process repeats. New and useful training examples - *i.e.* those that the classifier would otherwise classify incorrectly - can be added during this procedure because of the grouping in the unlabeled data. This intuition was introduced in [51];
CHAPTER 2. GROUP INDUCTION

What follows is the new formalization of this intuition.

Formally, group induction is the optimization problem

$$ \min_{H, y_g \in \{-1, 0, +1\}^g} \sum_i \ell(H(x_i), y_i) + \sum_g \sum_u \ell(H(x_{g,u}), y_g), \quad (2.1) $$

where $H$ is a classifier, $x \in \mathbb{R}^N$ is a descriptor, $y \in \{-1, 0, +1\}$ is a class label, and $\ell$ is a loss function. The two terms correspond to supervised training data and unlabeled groups, respectively, with $i$ ranging over the supervised data, $g$ ranging over all groups, and $u$ ranging over instances within a group.

Different machine learning models can be plugged in to this problem. For example, with logistic regression, $H(x) = w^T x$ and $\ell(H(x), y) = \log(1 + \exp(-yH(x)))$. Alternatively, with boosting, $H(x) = \sum_k h_k(x)$ is the strong classifier composed of many weak classifiers and $\ell(H(x), y) = \exp(-yH(x))$. Much of the following math remains the same or similar for different choices of $\ell$, but we will use boosting and exponential loss from this point on. Finally, we present only the binary classification
case; the extension to multiclass is straightforward.

The group induction problem as a whole is non-convex, but we can find a local minimum by alternating between an \textit{induction} phase and a \textit{training} phase, analogous to the expectation and maximization steps of EM [12].

In the induction phase, we hold $H$ fixed and solve for each hidden group label $y_g$. This problem is separable across groups, so we can consider in isolation the problem

$$\min_{y \in \{-1,0,+1\}} \sum_u \exp(-y H(x_u)), \tag{2.2}$$

where unnecessary indexing and constant terms have been stripped. Drawing inspiration from self-paced learning [24], the $y = 0$ label is allowed so that only unlabeled groups which we are confident in will actually be used for training. This problem has an analytical solution; we simply evaluate the objective function for each of the possible labels and choose the minimizer. This is significantly more conservative from the induction heuristic in [51], which proposed a threshold on average group confidence $\frac{1}{U} \sum_{u=1}^{U} H(x_u)$. See Figure 2.2 for a concrete example of the induction criteria.

The training phase entails holding the group labels fixed while training the classifier. In the generic case, this reduces to the standard training objective function of $\sum_i \ell(H(x_i), y_i)$, where $i$ ranges over all supervised instances as well as all instances in the inducted groups.

### 2.3.1 Worst-case classifier noise

In [51], one could adjust the induction speed with a threshold on average group confidence. Here, because the induction criteria has changed, that method no longer applies. Moreover, we would like to maintain the rigorousness of our optimization; if we were to simply decree that the induction phase must respect a threshold on average group confidence, we would no longer be optimizing the problem (2.1).

Instead, we can propose an alternative optimization problem which encodes the intuition of [51] in a rigorous way. The idea is simple: We assume that all classifier predictions are corrupted by a fixed amount of worst-case noise. The full optimization
CHAPTER 2. GROUP INDUCTION

problem becomes

\[
\begin{aligned}
& \text{minimize} & & \sum_i \exp(-y_i[H(x_i) + \epsilon_i]) \\
& \text{maximize} & & + \sum_g \sum_u \exp(-y_g[H(x_{g,u}) + \epsilon_{g,u}]).
\end{aligned}
\]

The optimization for the training phase reduces to

\[
\begin{aligned}
& \text{minimize} & & \sum_i \exp(-y_i[H(x_i) + \epsilon_{\max}]) + \sum_g \sum_u \exp(-y_g[H(x_{g,u}) + \epsilon_{\max}]),
\end{aligned}
\]

and the worst-case errors drop out as a constant factor. We are left with boosting training as usual.

During the induction phase, analogous to (2.2), we have

\[
\begin{aligned}
& \text{minimize} & & \sum_u \exp(-y[H(x_u) + \epsilon_{\max}]) \\
& \text{maximize} & & \sum_u \exp(-y[H(x_u) + \epsilon_u]).
\end{aligned}
\]

As before, we can simply evaluate this sum for each possible \(y\) and choose the smallest. Setting \(\epsilon_{\max} > 0\) allows us to make the induction more conservative, analogous to increasing the average group confidence threshold in [51].

2.3.2 Active learning and retrospection

In practice, it is often desirable to have a user provide occasional input to guide the system to a good local minimum. This amounts to adding new examples to the supervised term of (2.1). These new examples could be acquired from any of a number of active learning methods.

When new annotations arrive, we start a new optimization problem that is similar to the old one, but with additional terms of supervised data. The naïve warm-start for this new optimization problem is to simply use the previous classifier and group labels untouched.

However, this can have undesirable effects when using a highly discriminative classifier. Consider a case in which several false positive inductions have been made and
a user, watching the system make false positive classifications at runtime, provides new annotated examples to address the issue. Since the false positive inductions remain in the training set along with the new annotated examples, a highly discriminative classifier will learn to distinguish the two based on what seem to be minor and unnoticeable details.

What we actually want is to de-induct, *i.e.* set \( y_g = 0 \), just the false positive inductions. We refer to the process of using new annotated examples to de-induct certain groups as *retrospection*. Because we are choosing a warm start for a new optimization problem rather than actually taking a step during an optimization, we are free to apply heuristics that seem reasonable and effective.

One way to accomplish retrospection is to make use of the worst-case classifier noise formulation above: Do an induction step using \( \epsilon_{\text{max}} = \max_i [-y_i H(x_i)] \), where \( i \) ranges over the new annotated examples. This provides a lower bound on the true value of the worst-case noise, and the result is that low confidence groups will be de-inducted and high confidence groups will remain. A user annotation which reveals a high confidence false positive will result in a large \( \epsilon_{\text{max}} \) and therefore significant retrospection, whereas a low confidence false positive will result in only moderate retrospection. We also reset \( H \), as otherwise at the next induction phase, it will tend to re-induct the false positive groups we were intending to be rid of in the first place.

### 2.3.3 Unlabeled data streams

When considering streams of unlabeled data, we need to amend the optimization problem (2.1) somewhat, as the sum over the unlabeled groups grows continuously. In practice, we cannot store all unlabeled groups, so we resort to the approximation of keeping a fixed-size set of groups that can fit in memory. Specifically *which* groups is motivated directly by the math: Choose those that contribute the most to the training phase objective function, after constant terms (*i.e.* groups with \( y_g = 0 \)) are dropped. Intuitively, this corresponds to keeping the examples from which the classifier can learn the most.

We assume that all groups have a bounded number of instances and that unlabeled
data arrives in chunks with a bounded number of groups. During the induction phase, one new chunk is added to the unsupervised term, induction is run as described above, and then the least useful groups are dropped until the unsupervised term is equal to its maximum size. In this way, the group induction system has an upper bound on its memory footprint and thus can be expected to run indefinitely even as new unlabeled data streams in.

2.4 Track classification

2.4.1 Problem overview

In this section, we briefly review the track classification task introduced in [54] and addressed in [49, 51]. Track classification is defined as a sub-task of an object recognition pipeline which includes segmentation, tracking, and track classification. Objects are first segmented and tracked without semantic labels, then tracks are classified as a whole to determine semantic label. In our case, raw data from a Velodyne HDL-64E is clustered using simple obstacle detection and connected components analysis, and then these clusters are tracked using Kalman filters [28, 31]. This segmentation and tracking stage has no knowledge of semantic label, but it is known that all frames in a track are likely to share the same label. Track classifications are determined by taking the average of the frame classifications.

We make use of the Stanford Track Collection (STC) for our evaluation. This is a large dataset designed for evaluating on the track classification problem, containing about one million labeled frames (i.e. single views of objects) across about fourteen thousand tracks collected from natural street scenes. Example objects from the STC can be seen in Figure 2.3.

Segmentation and tracking errors are considered out of scope for the track classification task, as we need to avoid confounding these errors with classification errors. Thus, the test set does not include segmentation and tracking errors, though the unlabeled data we use for group induction does.
Figure 2.3: Example objects from the Stanford Track Collection [55]. This dataset contains tracks of objects segmented using a standard depth-segmentation algorithm used in autonomous vehicles [28]. Each track forms a group of instances, all of which are assumed to share the same hidden label.

2.4.2 Parametric boosting classifier

Many choices of classifier and loss function would likely be effective for group induction on this task. For example, we expect that one could probably achieve similar results using the standard online learning approach of stochastic gradient descent training of a logistic regression model\(^1\). For completeness, however, we present our particular choice, based on boosting. A brief introduction to boosting using notation similar to that of this paper can be found in [51], and further details can be found in [16], among many others.

We refer to our method as parametric boosting because it has the form of a boosting classifier, but draws from a fixed pool of possible weak classifiers. This results in a fixed number of parameters that must be stored no matter how many new weak classifiers are added.

The specialized boosting classifier of [51] used hyperspheres as weak classifiers.

\(^1\)with one caveat about descriptor space choice, discussed in Figure 2.4.
Training continuously, as we will do with group induction, entails adding new weak classifiers continuously, and at runtime one must compute distances to all hyperspheres to make a prediction. This is a significant impediment to lifelong learning because classification time grows as training proceeds.

To avoid this slowdown, we first specify a simple descriptor transformation. For each descriptor element in the original space, we define a one dimensional grid, or array of bins. Each bin represents a binary feature in the transformed space. See Figure 2.4 for a visualization.

Then, we choose the form of our weak classifiers to be \( h(x) = \alpha c(x) \), where \( c : \mathbb{R}^n \to \{0, 1\} \) is a simple test and \( \alpha \in \mathbb{R} \) is the response value for this weak classifier. In the case of [51], \( c \) encodes containment within a hypersphere, whereas in this paper it encodes containment in one of the aforementioned bins. The boosting training algorithm is similar to [51], but where we can now store one parameter per bin rather than per weak classifier. That is, multiple weak classifiers can live in the same bin, and their responses add.

The key property of this formulation is that we can express classification as a sum over the fixed number of bins rather than a sum over the growing number of weak classifiers. Formally, this is

\[
H(x) = \sum_{k=1}^{K} h_k(x) = \sum_{b=1}^{B} z_b c_b(x),
\]

where \( b \) is the bin index and \( z_b \) is the sum of the \( \alpha \)s for all weak classifiers that live in bin \( b \).

### 2.4.3 Descriptor spaces

We use the same 29 descriptor spaces as [51], all deriving from pointcloud data. These descriptors include oriented bounding box size, spin images, and HOG descriptors [11] computed on virtual orthographic camera images. This defines an aggregate descriptor space of \( \mathbb{R}^{4123} \). We then define two grids, one with 10 bins and one with 100 bins, across each element as described in the previous section. Bin width for a
Figure 2.4: Intuition for the grid feature transform we use. The bar plot shows histograms of positive and negative examples of cars vs descriptor element value. A naive application of the standard online learning technique of logistic regression assumes a linear relationship between each individual descriptor element and the log odds. That would mean assuming, for example, that as the height of an object increases, the log odds of being a car increases or decreases linearly; this is clearly wrong for our data. By instead having a binary variable per bin, we can learn that cars tend to live within a certain height range. The parametric boosting classifier of Section 2.4.2 builds on this feature transform by using weak classifiers that make predictions for a single bin.
descriptor element is determined by computing the minimum and maximum values on a large unlabeled dataset.

2.5 Experiments

2.5.1 Fully-supervised baseline

Before delving into group induction, we evaluate our parametric boosting classifier to ensure it can reach reasonable accuracy on a fully-supervised problem. Simultaneously, we provide an estimate of how many user-annotated examples are needed to reach a given level of accuracy.

In this experiment, we train using subsets of the \(\sim 8000\) tracks (\(\sim 700,000\) frames) in the STC training set and evaluate on the \(\sim 6000\) tracks in the STC test set. As in [51], the training set is broken down into hand-labeled data \(T_0\) and automatically-labeled negative examples \(B_0\). The latter dataset is collected by driving in places without pedestrians, bicyclists, or cars and automatically labeling all segmented objects as negative examples of each of the three classes. This dataset does not require hand-labeling of individual tracks and is typically quite easy to collect, so we do not count it towards the number of hand-labeled examples.

To estimate accuracy as a function of hand-labeled data, we provide \(T_0\) to the algorithm one logfile at a time, repeating the experiment four times with different logfile orderings. For each evaluation on a subset of \(T_0\), the full background dataset \(B_0\) is also provided. Results can be seen in Figure 2.5. The \(x\)-axis only counts the subset of \(T_0\).

The STC dataset was sampled uniformly from the objects seen by our vehicle, so the proportion of background objects is quite high. Predicting \(y = -1\) for all class problems results in accuracy of 81.8%. Previous work [51] achieved 98.7% correct, and the parametric boosting classifier of this paper gets to 98.3%. Thus, parametric boosting gives up a small amount of accuracy, but this is a small price to pay for constant runtime and thus the ability to run continuously.
Figure 2.5: Supervised accuracy of the parametric boosting classifier vs amount of training data provided. (Group induction is not used here.) Each line shows results for a different random ordering of the training data. Parametric boosting is similar to that used in previous work [51], but has the advantage of constant runtime - an essential property for lifelong learning. This experiment demonstrates that the new formulation can produce accuracy similar to the old method while delivering an asymptotic speed improvement. It also informs us of the number of training examples needed to achieve a given level of accuracy with supervised learning.
2.5.2 Group induction experiments

We evaluate group induction for autonomous driving on the STC test set. Because negative examples of cars, pedestrians, and bicyclists are so easy to collect, we simplify the group induction problem by only allowing $y_g \in \{0, 1\}$ and assuming that a large dataset of negative examples is provided. For this, we use $B_0$ from the previous section. We refer to this choice as single induction, as induction is only done for one of the binary labels; in contrast, dual induction is when one allows $y_g \in \{-1, 0, 1\}$, as in Section 2.3.

Our implementation otherwise follows that of Section 2.3, with $\epsilon_{\text{max}} = 0$ and using the retrospection strategy as described in Section 2.3.2. To ensure fixed memory footprint, we use the method described in Section 2.3.3 and limit group size to a maximum of 10. The latter is accomplished by simply breaking long unlabeled tracks into shorter ones. The group size of 10 is motivated by the use of exponential loss; long tracks are difficult to induct in the presence of segmentation and tracking noise, as visualized in Figure 2.2.

The number of unlabeled groups stored in memory is bounded by the approximation described in Section 2.3.3, with the maximum number of unlabeled groups set to 20,000. This number is motivated by the fact that $B_0$ contains about 150,000 frames, and so our number of inducted positive examples will be of the same order of magnitude.

We used $T_0$ stripped of its labels along with several additional unlabeled datasets for the unlabeled data, totaling 2.3M frames and 46GB on disk. To simulate an infinite data stream with this finite dataset, during each induction phase a random logfile was loaded and added as a new unlabeled chunk.

New user annotations were added by classifying logs, sorting by confidence, and visually inspecting the low-confidence tracks - a simple kind of active learning. Each run of active learning resulted in new user annotations, and each took on the order of one to ten minutes of user time. Total computation time of the experiment was about 2 days.

Experimental results are presented in Figure 2.6. The key results are that
• it is possible to achieve good accuracy (98.1%) using on the order of ten to one hundred user-annotated examples for each class problem, plus the automatically labeled background dataset $B_0$.

• supervised training on $B_0$ and all user-annotated tracks shown in Figure 2.6 produces only 88.6% accuracy; group induction is necessary.

• the baseline experiment of Figure 2.5 suggests that in a fully-supervised context, about 4000 user-annotated examples (in addition to $B_0$) would be necessary to reach the same level of accuracy.

• unlike [51], our method can learn continuously while maintaining constant classification speed.

• unlike [51], our method can adapt to small amounts of user feedback.

Moreover, the user annotations are dominated by 104 annotations of non-cars. In practice, this is an indication that one should simply use more automatically-labeled background data.

2.6 Conclusions

In summary, we have presented a new mathematical framework for semisupervised learning, applicable any time one has access to groups of unlabeled instances with shared hidden labels. We have shown group induction can dramatically reduce the amount of user-annotated data necessary and that, unlike previous work, it can make use of streams of unlabeled data and user feedback. Group induction is potentially a viable path to lifelong learning perception systems that are trainable by regular users rather than machine learning experts.

Several primary avenues of future work are clear. First, in our experiments we have made use of an automatically-labeled background dataset and used a single-induction implementation, i.e. the system only inducts positive examples. While the background dataset is often easy to collect, it would be better if it were not required.
Figure 2.6: Experimental results showing ability of group induction to adapt to user feedback and make use of large amounts of unlabeled data. Bar plots show user annotations, with positive examples in green and negative examples in gray. Notice, for example, the jump in recall when new positive examples of pedestrians are added. The retrospection strategy discussed in Section 2.3.2 is particularly evident in the final addition of negative examples to the car problem: Very confident false positive inductions were identified, leading to de-induction of most of the buffer. Performance then converges to a stable and accurate solution. Bicyclist and pedestrian plots are clipped to show detail in the early stages; they remain stable through the end. The final accuracy of group induction is 98.1%. In contrast, training on just the annotated examples from this run (i.e. without applying group induction) produces final accuracy of 88.6%. Predicting \( y = -1 \) for all class problems produces final accuracy of 81.8%. Amounts of user-annotated data in this figure can be fairly compared with those in the fully-supervised experiment of Figure 2.5.
Second, our segmentation and tracking solution is designed for street scenes, where objects tend to avoid under-segmentation with the environment. In less structured environments such as homes or offices, this kind of model-free segmentation and tracking is not generally available. Some initial work in this direction exists [50], but more work remains to be done before a sufficiently robust and real-time solution is available.
Chapter 3

Learning to Segment and Track

Abstract

We consider the problem of segmenting and tracking deformable objects in color video with depth (RGBD) data available from commodity sensors such as the Asus Xtion Pro Live or Microsoft Kinect. We frame this problem with very few assumptions - no prior object model, no stationary sensor, no prior 3D map - thus making a solution potentially useful for a large number of applications, including semi-supervised learning, 3D model capture, and object recognition.

Our approach makes use of a rich feature set, including local image appearance, depth discontinuities, optical flow, and surface normals to inform the segmentation decision in a conditional random field model. In contrast to previous work in this field, the proposed method learns how to best make use of these features from ground-truth segmented sequences. We provide qualitative and quantitative analyses which demonstrate substantial improvement over the state of the art.

This paper is an extended version of our previous work [50]. Building on this, we show that it is possible to achieve an order of magnitude speedup and thus real-time performance (\(\sim 20\)FPS) on a laptop computer by applying simple algorithmic optimizations to the original work. This speedup comes at only a minor cost in overall accuracy and thus makes this approach applicable to a broader range of tasks. We demonstrate one such task: real-time, online, interactive segmentation to efficiently collect training data for an off-the-shelf
Note to Practitioners — The original motivation for this work derives from object recognition in autonomous driving, where it is desirable to identify objects such as cars and bicyclists in natural street scenes. In previous work [49] it was shown that, so long as one can segment and track objects in advance of knowing what they are, it is possible to train accurate object detectors using a small number of hand-labeled examples combined with a large number of unlabeled examples. The key dependency of a model-free segmentation and tracking method was available because of the structure in the autonomous driving problem. This paper aims to make these techniques applicable in more general environments.

In the near term, the methods presented here can be used for real-time, online, interactive object segmentation. This can ease the process of collecting training data for existing object recognition systems used in automation today. In the long term, improved implementations could be an integral part of semi-supervised object recognition systems which require few hand-labeled training examples and can produce accurate recognition results.

3.1 Introduction

The availability of commodity depth sensors such as the Kinect opens the door for a number of new approaches to important problems in robot perception. In this paper, we consider the task of propagating an object segmentation mask through time. It is assumed that a single initial segmentation is given; in this work the initial segmentation is provided by human labeling, but this input could instead come from an automatic method depending on the application. We do not assume the presence of a pre-specified object model (e.g. as the Kinect person tracking system assumes a human skeleton), as that would preclude the system from segmenting and tracking arbitrary objects of interest. As such, this task falls into the category of model-free segmentation and tracking, i.e. no prior class model is assumed. Similarly, we do not assume the sensor is stationary or that a pre-built static environment map is
Figure 3.1: Definition of the model-free segmentation and tracking task addressed in this paper. Given an initial seed labeling (first frame; cat on a coffee table), the goal is to produce a segmentation mask of the object over time (subsequent frames) without a pre-trained object model. Foreground points are shown bold and colored, background small and gray.

available. As “model-free segmentation and tracking” is somewhat unwieldy, we will sometimes refer to this task as STRGEN (sturgeon), for “segmentation and tracking of generic objects”.

3.1.1 Example use cases and long-term vision

There are several reasons a complete solution to the STRGEN task would be useful in robotics, computer vision, and graphics. For example, this would make possible a generic and natural object selection interface which could be used for 3D model capture in unstructured environments. In the following, however, we will focus primarily on the area of object recognition. In the first section we describe how our method can already be a useful part of a standard object detection pipeline; the following sections then outline potential future uses.

Training data collection for standard object detectors

Much work has been put into object detection methods using RGBD sensors. While off-the-shelf object detection algorithms such as LINE-MOD [20] are often fast and
CHAPTER 3. LEARNING TO SEGMENT AND TRACK

effective at runtime, training data collection remains a hassle. Typical approaches include turntables and crowdsourcing. While the former allows for high-quality training examples for a full 360° range of views, it also involves an appropriate rig, is limited to objects that can be easily placed in a rig, and must be repeatedly reconfigured for different perspectives. The latter allows for unstructured environments and avoids the physical setup annoyances but also requires a well-designed web application and a data processing framework robust to noisy and incompatible responses. This approach also could be inapplicable to certain data sources due to privacy concerns. Other data labeling approaches, such as iteratively placing objects in a scene and subtracting a known background, are rare as they often have disadvantages such as being slower to collect significant quantities of training data.

Using a STRGEN algorithm for the collection of training data avoids these disadvantages. With a handheld depth sensor and laptop computer, training data can be collected in unstructured environments by taking sparse labeling hints from the user and generating segmentations over time, providing new training examples with different object poses, view angles, lighting conditions, and so on, with little additional human effort. Labeled, segmented objects can be collected in real time while the raw sensor data is recorded. A prototype of such a system is shown in Figure 3.2; see also Section 3.4.3 for an experiment using a similar concept.

Track classification

Currently, most object recognition methods fall into semantic segmentation or sliding window categories. When a solution to the STRGEN problem is available, however, it is possible to use a different approach, exemplified in [54]. In this approach, objects are segmented and tracked over time without a semantic label and the track as a whole is classified online as new segmented frames stream in; we refer to this object recognition problem breakdown as STAC, for segmentation, tracking, and classification. As shown in [54], the STAC approach significantly improves classification accuracy because of its ability to use multiple views over time and to use track descriptors such as average object speed or maximum angular velocity. In [54], model-free segmentation and tracking was available due to the specific circumstances of autonomous driving.
Figure 3.2: Example usage of model-free segmentation and tracking algorithm for object detector training set collection. Initially, no object is being tracked. A user provides sparse foreground and background hints as the RGBD data streams in. Segmented objects are shown in red.

that are not generally applicable. A solution to the general model-free segmentation and tracking problem, however, would enable this approach in less structured environments.

Tracking-based semi-supervised learning

A solution to the STAC problem opens the door to a simple and effective method of semi-supervised learning in which a large number of unlabeled tracks of objects are used in conjunction with a small number of hand-labeled tracks to learn an accurate classifier. This method, known as tracking-based semi-supervised learning (TBSSL), was demonstrated in the autonomous driving context using laser range finders to learn to accurately recognize pedestrians, bicyclists, and cars versus other distractor objects in the environment using very few hand-labeled training examples [49]. However, as with [54], model-free segmentation and tracking was more or less freely available because objects on streets generally avoid collision and thus remain depth-segmentable.
using simple algorithms. To extend tracking-based semi-supervised learning to the 
more general case in which objects are not easily depth-segmentable, a more general 
solution to the STRGEN problem is required.

Long-term vision

The primary long-term motivation for this work is to enable object recognition sys-
tems that produce high-quality results, yet are trainable and usable by regular people 
using only modest amounts of effort. The success of STAC and TBSSL, given model-
free segmentation and tracking, hint that this may be possible. One option is to (A) 
collect a small number of training examples using a natural interface such as that 
of Figure 3.2, (B) apply TBSSL to learn a high-quality classifier from large quan-
tities of unlabeled data, and (C) run STAC to recognize objects using the learned 
classifier. The primary missing component in this system or variations thereof is a 
high-accuracy and real-time solution to the general STRGEN problem.

3.2 Previous work

While there is much previous work on tracking in general, our problem’s online nature, 
output of segmentation masks rather than bounding boxes, and lack of simplifying as-
sumptions restrict the directly-related previous work substantially. The most similar 
and recent work to ours is HoughTrack [18], which uses Hough Forests and GrabCut 
to segment consecutive frames. We provide a quantitative comparison to HoughTrack 
in Section 3.4.1. Older previous work on this problem include [5] and their multi-
object extension [6], which use a generative model and level sets; [25], which tracks 
local object patches using Basin Hopping Monte Carlo sampling; and [37], which uses 
a conditional random field model and loopy belief propagation. All these works limit 
themselves to only a small number of features and do not use learning methods to find 
the best general-purpose tracker. Additionally, none of these works consider depth 
information.

We now briefly review previous work on related problems.
Interactive segmentation - Human-assisted object segmentation has been the subject of much work including interactive graph cuts [8], GrabCut [38], and Video SnapCut [3]. These methods are largely intended for graphics or assisted-labeling applications as they require a human in the loop. This paper addresses automatic segmentation with the exception of the application example in Section 3.4.3.

Bounding box tracking - Discriminative tracking [19, 46, 23] addresses the online, model-free tracking task, but where the goal is to track arbitrary objects in a bounding box rather than provide a segmentation mask.

Rigid object tracking - Rigid object tracking using depth information, such as the open source method in PCL [39], addresses a similar but simpler problem, as it is not designed to work with deformable objects.

Offline methods - The work of [9] takes as input segmentations for the first and last frames in a sequence, rather than just the first frame. While [60] takes as input only the segmentation of the first frame, they construct a CRF on the entire video sequence, with CRF labels corresponding to object flows. Here we are interested in a method that can update as new data streams in, thus making it applicable for robotics or other online vision tasks.

Background subtraction - Background subtraction approaches can greatly simplify the segmentation task. For example, [1] assumes a stationary camera to produce fine-grained segmentation masks for multiple objects. With depth sensors, background subtraction methods can also operate while the sensor is in motion: Google’s autonomous car project uses pre-built 3D static environment maps to subtract away all 3D points except the moving obstacles [62]. This enables simple depth segmentation to be effective as long as moving objects do not touch, but assumes a high-precision localization system as well as a pre-learned static environment map.

Model-based tracking - For some tasks, it is appropriate to model a specific object that is to be tracked, either with an explicit 3D model as in [35] or with pre-trained statistical models that are specific to the particular object class. Examples of the latter approach include the human pose tracking of the Kinect [43] and model-based car tracking in laser data for autonomous driving [34].

It is worth noting that the line between model-free and model-based tracking is not
sharp. Because several of our intended applications revolve around training detectors for previously-unseen objects, we restrict ourselves to making use of object models that are learned on the fly. It is entirely conceivable, however, that one could plug in a detailed pre-trained object model to our framework to produce higher accuracy results for that particular object.

Similarly, using a SLAM system to generate a known background map would improve the segmentation process by allowing for static background subtraction even with a moving sensor. However, a SLAM system would prevent the segmentation algorithm from working in unmapped areas or on objects that are part of the static environment, so we consider only background models that are learned on the fly and are relative to the chosen foreground object.

3.3 Approach

There are a large number of possible cues that could inform a segmentation and tracking algorithm. Optical flow, image appearance, 3D structure, depth discontinuities, color discontinuities, etc., all provide potentially useful information. For example:

- An optical flow vector implies that the label of the source pixel at time $t - 1$ is likely to propagate to that of the destination pixel at time $t$.

- Pixels with similar color and texture to previous foreground examples are likely to be foreground.

- The shape of the object in the previous frame is likely to be similar to the shape of the object in the next frame.

- Points nearby in 3D space are likely to share the same label.

- Nearby points with similar colors are likely to share the same label.

Previous work generally focuses on a few particular features; here, we advocate the use of a large number of features. The above intuitions can be readily encoded by node and edge potentials in a conditional random field model. However, this
additional complexity introduces a new problem: how much importance should each feature be assigned? While using a small number of features permits their weights to be selected by hand or tested with cross validation, this quickly becomes impractical as the number of features increases.

The margin-maximizing approach of structural SVMs [48, 61], adapted to use graph cuts for vision in [47], provides a solution to learning in these scenarios. Intuitively, this method entails running MAP inference, and then adding constraints to an optimization problem which assert that the margin between the ground truth labeling and the generated (and generally incorrect) labeling should be as large as possible. The application of this approach will be discussed in detail in Section 3.3.2.

3.3.1 Conditional random fields and inference

A conditional random field is an undirected graphical model that can make use of rich feature sets and produce locally-consistent predictions. It aims to spend modeling power on only the distribution of the target variables given the observed variables. In particular, the conditional random field takes the form

\[
P(y|x) = \frac{1}{Z(x)} \exp(-E(y, x)),
\]

where \(Z\) is the normalizer or partition function, \(y \in \{-1, +1\}^n\) is the segmentation for an image with \(n\) pixels, and \(x\) is a set of features defined for the RGBD frame, to be detailed later. The energy function \(E\) contains the features that encode various intuitions about what labels individual pixels should take and which pixels should share the same labels. In particular, the energy function is defined as

\[
E(y, x) = \sum_{i \in \Phi_\nu} w_i \sum_{j \in \nu_i} \phi_j^{(i)}(y, x) + \sum_{i \in \Phi_\epsilon} w_i \sum_{(j,k) \in N_i} \phi_{jk}^{(i)}(y, x).
\]

Here, \(\Phi_\nu\) is the set of node potential indices (i.e. one for each type of node potential such as local image appearance or 3D structure alignment), \(\nu_i\) is the set of all node indices for this potential type (normally one per pixel), and \(\phi_j^{(i)}\) is the node potential of type \(i\) at pixel \(j\). Similarly, \(\Phi_\epsilon\) is the set of edge potential indices, \(N_i\) is the
neighborhood system for edge potential type \( i \) (normally pairs of neighboring pixels), and \( \phi_{jk}^{(i)} \) is the edge potential between pixels \( j \) and \( k \) for edge potentials of type \( i \). Thus, the weights apply at the feature-type level, i.e. \( w_i \) describes how important the \( i \)th feature is.

The goal of MAP inference in this model is to choose the most likely segmentation \( y \) given the features \( x \) by solving

\[
\max_y \, \Pr(y|x) = \min_y \, E(y, x).
\]

During inference, the weights \( w \) remain fixed. The solution to this problem can be efficiently computed for \( y \in \{-1, +1\}^n \) and submodular energy function \( E \) using graph cuts [8].

### 3.3.2 Learning

Given a dataset of \((y_m, x_m)\) for \( m = 1 \ldots M \), the goal of CRF learning is to choose the weights \( w \) that will result in the lowest test error. While it would be desirable to
learn the weights directly using the maximum likelihood approach

\[
\max_w \prod_m P(y_m | x_m),
\]

(3.6)

this is not possible to do exactly because of the presence of the partition function \( Z(x) = \sum_y \exp(-E(y, x)) \) in the gradient. This function sums over all \( 2^n \) segmentations of an image with \( n \) pixels.

Fortunately, there is an alternative approach known as the structural support vector machine [48, 61, 47], which we now briefly review. Solving the margin maximizing optimization problem of (3.4) would result in a good solution. Here, \( C \) is a constant, \( \Delta \) is a loss function, \( \xi_m \) is a slack variable for training example \( m \), and \( \mathcal{Y} \) is the set of all possible labelings of an image. This problem, too, is intractable because it has exponentially many constraints. However, one can iteratively build a small, greedy approximation to the exponential set of constraints such that the resulting weights \( w \) are good.

Further, one can transform (3.4), known as the \( n \)-slack formulation, into the equivalent problem (3.5) as described in [21]. As before, while the number of constraints is exponential, one can build a small, greedy approximation that produces good results. Known as the 1-slack formulation, this problem is equivalent and can be solved much more efficiently than (3.4), often by one or two orders of magnitude.

The goal is to learn the weights that will best segment an entire sequence given a single seed frame. However, the segmenter operates on a single frame at a time, so our training dataset must be in the form of \((y, x)\) pairs. As will be discussed in Section 3.3.3, some of the features are stateful, i.e., they depend on previous segmentations. This presents a minor complication. At training time, we do not have access to a good choice of weights \( w \), but the features depend on previous segmentations which in turn depend on the weights. To resolve this, we adopt the simple strategy of generating features assuming that previous segmentations were equal to ground truth. This dataset generation process is specified in Algorithm 1.

The structural SVM solver is detailed in Algorithm 2. Because the graph cuts solver assumes a submodular energy function, non-negative edge weights \( w_i \) for all
3.3.3 Energy function terms

We now review the particular choice of energy function (3.2) that we use in our implementation. The particular choice of features can be modified or added to without substantially changing the underlying framework. See Figure 3.3 for visualizations. All node potentials are designed to be constrained to $[-1, +1]$, and all edge potentials to $[-1, 0]$. While not strictly necessary, this aids in interpreting the weights learned by the structural SVM.
Algorithm 2 Structural SVM for learning to segment and track

\( \mathcal{D} \) is a set of training examples \((y, x)\), formed as described in Algorithm 1. 
\( C \) and \( \epsilon \) are constants, chosen by cross validation.

\( \mathcal{W} \leftarrow \emptyset \)
repeat
repeat
    Update the parameters \( w \) to maximize the margin.

    minimize \( w, \xi \) \( \frac{1}{2} ||w||^2 + C\xi \)
    subject to \( w \geq 0, \quad \xi \geq 0 \)

    \( \frac{1}{M} \sum_{m=1}^{M} E(\hat{y}_m, x_m) - E(y_m, x_m) \geq \frac{1}{M} \sum_{m=1}^{M} \Delta(y_m, \hat{y}_m) - \xi \)

    \( \forall (\hat{y}_1, ..., \hat{y}_M) \in \mathcal{W} \)

    for \( (y_m, x_m) \in \mathcal{D} \) do
        Find the MAP assignment using graph cuts.
        \( \hat{y}_m \leftarrow \arg\min_y E(y, x_m) \)
    end for

    \( \mathcal{W} \leftarrow \mathcal{W} \cup \{(\hat{y}_1, ..., \hat{y}_M)\} \)
until \( \frac{1}{M} \sum_{m=1}^{M} \Delta(y_m, \hat{y}_m) - E(\hat{y}_m, x_m) + E(y_m, x_m) \leq \xi + \epsilon \)
Node potentials

Node potentials capture several aspects of shape, appearance, and motion. All take the form

$$\phi_j(y, x) = \begin{cases} a_j & \text{if } y_j = 1 \\ b_j & \text{if } y_j = -1, \end{cases}$$

(3.7)

where $a_j$ and $b_j$ are functions of the data $x$, and $y_j = 1$ indicates that pixel $j$ has been assigned to the foreground.

**Seed labels** - The first frame of every sequence provides a foreground/background segmentation. Seed label node potentials can also be used after the first frame as additional segmentation hints, used in this paper only for construction of the ground truth dataset and for the interactive experiment of Section 3.4.3. Each pixel in the image can be labeled by the user as foreground or background, or left unlabeled. Concretely, let $s_j \in \{-1, 0, +1\}$ be the seed labels for each pixel. Then, the potential is defined by

$$\phi_j(y, x) = \begin{cases} -1 & \text{if } y_j = s_j \\ 0 & \text{otherwise}. \end{cases}$$

**Optical flow** - Pixels in the current frame with optical flow vectors from the previous frame are likely to have the label of the originating pixel. If no optical flow vector terminates at pixel $j$ in the current frame, let $f_j = 0$. Otherwise, let $f_j$ be equal to the label of the originating pixel from the previous frame. Then, the potential is defined by

$$\phi_j(y, x) = \begin{cases} -1 & \text{if } y_j = f_j \\ 0 & \text{otherwise}. \end{cases}$$

**Frame alignment bilateral filter** - Optical flow provides correspondences from one frame to the next; after aligning these frames, a bilateral filter given 3D position and RGB values is used to blur the labels from the previous frame into the current frame, respecting color and depth boundaries.
Figure 3.3: Visualizations of selected edge and node potentials in the CRF. Edge potentials are zoomed in to show fine structure. Strong edge potentials are darker, while weaker edge potentials fade to original image color. Node potentials expressing a preference for foreground are shown in red, for background in green, and for neither in black. Top row: original image, canny edge potentials, color distance, depth edge potentials. Bottom row: ICP, frame alignment bilateral filter, optical flow, and patch classifier node potentials. Best viewed in color.

Specifically, alignment between the two frames is found using a RANSAC method, with 3D correspondences given by optical flow vectors. Flow vectors that originate or terminate on a depth edge are rejected, as their 3D location is often unstable. After alignment, we find the set of neighboring points \( N \) from the previous frame within a given radius of the query point, then compute the value

\[
  z_j = \sum_{k \in N} y'_k \exp \left( -\frac{||c_j - c_k||_2}{\sigma_c} - \frac{||p_j - p_k||_2}{\sigma_d} \right),
\]

where \( y'_k \) is the \( \{-1, +1\} \) label of point \( k \) in the previous segmentation, \( c_j \) is the RGB value of pixel \( j \), and \( p_j \) is the 3D location of pixel \( j \). The two \( \sigma \)s are bandwidth parameters, chosen by hand. The value \( z_j \) is essentially that computed by a bilateral filter, but without the normalization. This lack of normalization prevents a point with just a few distant neighbors from being assigned an energy just as strong as a point with many close neighbors. Referring to (3.7), the final energy assignment is made by setting \( b_j = 0 \) and \( a_j = 1 - 2/(1 + e^{-z_j}) \).
Patch classifiers - Two different parameterizations of a random fern patch classifier similar to [32] are trained online and used to make pixel-wise predictions based on local color and intensity information.

A random fern classifier is a semi-naive Bayes method in which random sets of bit features are chosen (“ferns”), and each possible assignment to the bit features in the fern defines a bin. Each of these bins maintains a probability estimate by simple counting. The final probability estimate is arrived at by probabilistically combining the outputs of many different randomly generated ferns. See [32] for details. We use 50 randomly generated ferns, each of which have 12 randomly generated features.

Each bit feature in a fern is generated by very simple functions of the data contained in the image patch. For example, one of the bit features we use is defined by

$$z = \begin{cases} 1 & \text{if } I_{p_0} < I_{p_1} \\ 0 & \text{otherwise} \end{cases}$$

where $p_0$ and $p_1$ are two pre-determined points in the image patch, and $I_p$ is the intensity of the camera image at point $p$. Other bit features include color comparisons and Haar wavelet comparisons, efficiently computed using the integral image.

At each new frame, the classifier is updated using the graph cuts segmentation output from the previous frame as training data, then is used to compute a probability of foreground estimate for every pixel in the current frame. The value $a_j$ is set to the probability of background, and $b_j$ to probability of foreground.

Distance from previous foreground - Points very far away from the object’s position in the previous frame are unlikely to be foreground.

After alignment of the current frame to the previous frame using the same optical-flow-driven method as the bilateral term, we compute the distance $d$ from the point $j$ in the current frame to the closest foreground point in the previous frame. The final potentials are set to $a_j = 0$ and $b_j = \exp(-d/\sigma) - 1$.

ICP - The iterative closest point algorithm is used to fit the foreground object’s 3D points (and nearby background points) from the previous frame to the current frame. If the alignment has a sufficient number of inliers, points in the current frame
are given node potentials similar to those of the bilateral node potential, but where the ICP-fit points are used instead of the entire previous frame.

**Prior term** - To express a prior on background, we add a term for which $a_j = 0$ and $b_j = -1$.

**Edge potentials**

Edge potentials capture the intuition that the foreground/background boundary is likely to be at an image gradient, color change, depth discontinuity, or surface normal change. All take the form

$$
\phi_{jk}(y, x) = \begin{cases} 
  a_{jk} & \text{if } y_j = y_k \\
  0 & \text{otherwise},
\end{cases}
$$

where $a_{jk}$ is a function of the data $x$. All edges are between neighboring points in the image.

**Canny edges** - All neighboring pixels are connected by an edge potential except those cut by canny edges. Concretely,

$$
a_{jk} = \begin{cases} 
  -1 & \text{if neither pixel lies on a Canny edge} \\
  0 & \text{otherwise}.
\end{cases}
$$

**Color distance** - Edge weights are assigned based on Euclidean distance between neighboring RGB values; $a_{jk} = -\exp(-||c_j - c_k||/\sigma)$, where $c_j$ and $c_k$ are the RGB values of the two pixels and $\sigma$ is a bandwidth parameter, chosen by hand.

**3D distance** - Points nearby in 3D are likely to share the same label, especially if they lie in the same plane. Out-of-plane distance changes are penalized more heavily than in-plane distance changes. Specifically,

$$
a_{jk} = -\exp \left( -\frac{||p_j - p_k||}{\sigma_d} - \frac{||p_j - p_k||}{\sigma_n} \right),
$$

where $p_j$ and $p_k$ are the neighboring 3D points, $n_k$ is the surface normal at point $p_k$, and the $\sigma$s are bandwidth parameters chosen by hand.
CHAPTER 3. LEARNING TO SEGMENT AND TRACK

Figure 3.4: While image appearance and depth information may sometimes provide little information about an object boundary, a change in surface normals can be informative. Surface normals are shown as small dark lines, seen from two angles to aid in depth perception.

**Surface normals** - Neighboring points that share the same surface normal are likely to have the same label. See Figure 3.4. We use $a_{jk} = -\exp(-\theta/\sigma)$, where $\theta$ is the angle between the two normals and $\sigma$ is a bandwidth parameter chosen by hand.

**Edge potential products** - Several combinations of the above are also provided, taking the form $a_{jk} = -|\prod_i a^{(i)}_{jk}|$, where $a^{(i)}_{jk}$ is the value from one of the above edge potentials. Intuitively, this encodes an edge potential that is strong (*i.e.* favors label equality) if all of the component edge potentials are strong.

### 3.4 Experiments

We provide a quantitative analysis to demonstrate improvement over the state of the art and a qualitative discussion of the strengths and weaknesses of the current implementation. All experiments use QQVGA (*i.e.* $160 \times 120$) RGBD data. The structural SVM of Algorithm 2 was implemented with standard interior point solver techniques. The graph cuts solver of [8] was used for inference.
CHAPTER 3. LEARNING TO SEGMENT AND TRACK

Figure 3.5: Comparison of our method with the most similar state-of-the-art work. Hamming loss is absolute number of pixels wrong; normalized accuracy takes into account object size and has a maximum of one. Total error reduction is about 65%.

3.4.1 Quantitative analysis

As there is no work we are aware of which does segmentation and tracking of non-rigid objects using depth data or learning of the best general purpose segmenter, we compare against HoughTrack, discussed in Section 3.2. We used the implementation that accompanies [18]. To ensure a fair comparison, we modified the HoughTrack implementation to be initialized with a segmentation mask rather than a bounding box.

No RGBD segmentation and tracking dataset currently exists, so we generated one containing 28 fully-labeled sequences with a total of about 4000 frames. Objects include, for example, a sheet of paper on a desk, cat, jacket, mug, laptop, and hat. In general, the dataset includes objects with different levels of rigidity, texture, and depth change from the surroundings. Ground truth was generated by hand labeling, assisted with an interactive version of the segmenter, similar to the interactive graph cuts work of [8].

The dataset was split into a training set of about 500 frames over 10 sequences and a testing set of about 3500 frames over 18 sequences. Training of our method was run once on the training set and the resulting segmenter was used to produce
results for all testing sequences. HoughTrack has no training stage, and thus did not use the training set. Testing results were produced for both by providing an initial segmentation for the first frame of each sequence, and all further frames were segmented without additional input.

Individual frames are evaluated with two different metrics. Hamming loss, or total number of pixels wrong, is simple and direct but ignores the size of the object; a Hamming loss of twenty could be negligible or a significant fraction of the object in question. To correct for this, we also report \textit{normalized accuracy}, which is 1 if all pixels in the frame are correctly labeled and 0 if the number of incorrectly labeled pixels equals or exceeds the number of pixels in the foreground object. More precisely, normalized accuracy is $1 - \min\left(1, \frac{\text{num\_wrong}}{\text{num\_fg}}\right)$. Sequence results are reported as the average of these values over all frames.

Overall, our method demonstrates a normalized error reduction of about 65% compared to the state-of-the-art. Detailed results can be seen in Figure 3.5. Training of our method takes a total of about 10 minutes and is done only once. At runtime, HoughTrack’s implementation takes about 200ms per frame, whereas our method’s implementation takes about 1200ms per frame.\footnote{Both implementations were compiled with optimizations turned on, run on the same machine, and given one thread.} Fortunately, it is possible to dramatically speed up our method; see Section 3.4.2.

Finally, one might wonder if it is the addition of depth data alone that provides the improvement relative to the baseline. To evaluate this, we removed all node and edge potentials that made use of depth data and added a bilateral filter that is analogous to the one described in Section 3.3.3 but uses only image data. Results are shown in Figure 3.6. Total error reduction is about 40% relative to HoughTrack.

### 3.4.2 Qualitative analysis

**Strengths**

Our results demonstrate that this method can be effective even in sequences which include significant non-rigid object transformations, occlusion, and a lack of visually
distinguishing appearance. As seen in the illustration in Figure 3.3, depth provides a very powerful cue as to what neighboring points are likely to share the same label. In addition to depth discontinuities, surface normals (Figure 3.4) can provide useful information about object boundaries, enabling the segmentation and tracking of objects for which using visual appearance alone would be extremely challenging, such as that of Figure 3.7. While this sequence would be relatively easy for off-the-shelf 3D rigid object trackers such as that in PCL [39], the tracking of deformable objects such as cats and humans would not be possible.

Our approach is general enough to handle both objects with few visually distinguishing characteristics and objects which are non-rigid; see Figure 3.8 for examples. In particular, the interface between the cat and the girl in sequences (A) and (B), and the interface between the cat and the bag in sequence (C) are maintained correctly. The method also works for objects without significant depth, such as the paper on the desk in sequence (G), and can recover from heavy occlusion as in sequence (F). The hat in sequence (H) undergoes deformation and heavy RGB blurring due to darkness of the environment, yet is tracked correctly. Sequence (D) shows the tracking of a piece of bread with Nutella while it is being eaten.

Figure 3.6: Comparison of our method with the most similar state-of-the-art work when dropping node and edge potentials that make use of depth data. Total error reduction is about 40%.
CHAPTER 3. LEARNING TO SEGMENT AND TRACK

Weaknesses and future work

As is typical in tracking tasks, stability versus permissiveness is a tradeoff. In some cases, it is not well defined whether a recently-unoccluded, disconnected set of points near the foreground object should be part of the foreground or part of the background; this is a common cause of errors in the current implementation. An example of this occurs in sequence (E), when the cat’s head becomes self-occluded, then unoccluded, and the system cannot determine that these points should be assigned to foreground. The error is then propagated. Similarly, the girl’s far arm in sequence (A) is lost after being occluded and then unoccluded. Rapid motion appears to exacerbate this problem. In general, it appears that more work is required to produce acceptable results on objects such as entire people which have many rapidly self-occluding and self-unoccluding articulated parts. While partial occlusion can be handled, as in sequence (F), the current implementation has no facility for re-acquisition if the target is completely lost. Finally, thin parts of objects are often problematic, as it seems the edge potentials are not well connected enough in these areas.

Fortunately, it is likely that improvements to node and edge potentials can resolve these limitations. Using longer range edges (rather than a simple 2D grid connecting neighbors only) could improve performance on thin objects and quickly moving
objects, and improvements in the image-appearance-based patch classifier could result in better segmentations of objects that self-occlude parts frequently. A node potential based on LINE-MOD or a discriminative tracker such as [23] could solve the re-acquisition problem, and could be run on object parts rather than the object as a whole to enable usage on non-rigid objects. This could be particularly helpful for parts of objects like arms which become briefly occluded and then unoccluded, but are disconnected in the 3D pointcloud from the rest of the person.

The feature-rich representation we use presents a challenge and an opportunity. There are a large number of parameters in the computation pipeline which cannot be learned via the structural SVM (e.g. the $\sigma$’s discussed in Section 3.3.3). There is also substantial freedom in the structure of the computation pipeline. Choosing the structure and the parameters by hand (as we have done here) is possible, but onerous; there is an opportunity to learn these in a way that maximizes accuracy while respecting timing constraints.

Finally, the method we have presented learns how to combine a diverse set of features for the best possible general purpose segmenter. There is a significant opportunity to make this method learn on the fly the best weighting of the features for a particular object. For example, it could be learned online that the surface normal edge potentials are not very helpful while tracking a poster on the wall.

Real-time performance

So far, we have discussed a relatively slow implementation of the segmentation and tracking algorithm. While this version is still useful for tasks such as offline object labeling or model building, a real-time algorithm is desirable. This is particularly important in realizing some of the goals discussed in Section 3.1.1, such as using the algorithm as a component of tracking-based semi-supervised learning. In this section, we show how to achieve real-time performance with only a very small loss in segmentation accuracy. Feature computation is by far the bottleneck, so we can focus our efforts entirely on optimizing this part of the pipeline.

We make use of two key ideas in achieving real-time performance. First, one need only compute features and feature pre-requisites at the boundaries of the target
Figure 3.8: Visualization of results. The first frame in each sequence (far left) is the seed frame. Foreground points are shown in bold and color while background points are shown small and gray. Best viewed on-screen.
object. Intuitively, there is no point in computing surface normals in the upper right corner of the RGBD image when the object of interest is in the lower left. Second, it is not necessary to compute expensive node potentials at every pixel in the image because the edge potentials in the CRF will tend to smooth out the final segmentation. Examples comparing the original and real-time methods can be seen in Figure 3.9.

Concretely, at each timestep, a boundary mask is computed by dilating the background and foreground masks from the previous segmentation and taking their intersection; this provides a mask that highlights only the edges of the object. Then, all following steps in the computation pipeline compute only inside the boundary mask. Node and edge potentials described in Section 3.3.3 are thus not computed in non-boundary regions. Instead, we force all of these pixels to take the label they had from the previous segmentation. Further, inside the boundary mask, downsampling is applied to the patch classifier, bilateral, and ICP node potentials.

To show that this optimization does not adversely affect segmentation performance, we ran ten iterations comparing the optimized segmentation algorithm versus the original algorithm using the same experimental setup as Section 3.4.1. Quantitative results showing overall segmentation accuracy and runtimes are shown in Figure 3.10. Experiments were run using a multi-threaded implementation on a laptop with an i7-3820QM processor. With use of the boundary mask and 50% downsampling, it is possible to run the algorithm at ~20FPS without sacrificing too much accuracy.

These results were all computed using QQVGA images. While this may seem unreasonably small, it is actually acceptable for tasks that do not require high resolution boundaries. For example, you might want to use the segmentation and tracking algorithm to collect training data for an object detector. The detector might require VGA (i.e. $640 \times 480$) data to produce good results, but computing the segmentation at QQVGA and scaling it up to the VGA image is acceptable.

Even at VGA, it is conceivable that real-time operation could be obtained with significant improvement to the feature set and code optimizations. As shown in Figure 3.11, the graph cuts solver does not preclude real-time segmentation.
Figure 3.9: Visualization of the original method versus the real-time version. Node potential and edge potential visualizations follow the same conventions as in Figure 3.3. Best viewed in color.
Figure 3.10: Test set normalized loss and average per-frame segmentation time compared with different levels of algorithmic optimization as discussed in Section 3.4.2. Thirteen runs of each condition were made; error bars and annotations show one standard deviation for this dataset. While a 75% downsampling induces too much error to be acceptable, 50% downsampling appears to be a good trade-off for many applications. Timing results for the original method differ from those presented in Section 3.4.1 because multithreading is enabled here.

3.4.3 Use-case example: training object detectors

In this section, we describe the application of our segmentation and tracking method to the problem of data labeling in supervised object detection systems such as LINE-MOD. Such systems rely on well-segmented training data typically obtained from either meticulously rigged turntables or elaborate crowd-sourcing applications, as discussed in Section 3.1.1. In contrast, by slightly modifying the STRGEN algorithm to incorporate the user’s segmentation hints as node potentials (discussed in Section 3.3.3) throughout the sequence, we provide an interactive, real-time means to accurately collect segmented training examples with minimal environmental control. Concretely, the original algorithm generated a node potential for seed labels on the first frame; here we simply allow these node potentials to be generated for subsequent frames as well.

The process, illustrated in Figure 3.12, only assumes the user is equipped with a laptop and an RGBD depth sensor. For our experiments, we attached an Asus Xtion Pro Live to the laptop of Section 3.4.2. To begin, the user starts recording the RGBD sequence and provides some rough initial foreground and background labels. These are interpreted as node potentials by the STRGEN algorithm and the
segmentation is propagated as additional frames stream in. In the case of errant segmentations, the user provides in real time additional foreground and background labels, and the STRGEN algorithm incorporates these as with the initial hints. At the end of the process, the user is left with a collection of segmented objects instances. Not all segmentations are perfect, but almost all bad segmentations are followed by a user correction. Thus, we use the five second rule: All segmentations that were generated less than about five seconds before any user-generated segmentation hint are ignored and the rest are added to the training set. At this point, it would probably be beneficial to generate many artificial viewpoint variations and add these to the training set, as well; this step is bypassed for this small experiment.

In Figure 3.13, we demonstrate how these segmented examples can be used to construct an object detector. We offer a small example in which we aim to detect a couch pillow in fifteen diverse scenes. Our method does not rely on any particular detection approach, but since we are interested in real-time detection, we feed our labeled pillow examples into the LINE-MOD template-matching framework described in [20]. At training time, LINE-MOD uses the depth and image cues to build templates (Figure 3.13, upper-left). At testing time, LINE-MOD searches for those templates in the previously unseen images and identifies any “detection” as a template match with a response exceeding some threshold (0.75 in our experiment). As can be seen in the illustration, we detect all objects while avoiding any false positives using a training set collected online with the STRGEN implementation.

3.5 Conclusions

We have presented a novel method of segmenting and tracking deformable objects in RGBD, making minimal assumptions about the input data. We have shown this method makes a significant quantitative improvement over the most similar state-of-the-art work in segmentation and tracking of non-rigid objects, that real-time performance is feasible, and that an online and interactive version of the algorithm is suitable for collecting training examples of objects in unstructured environments. A complete solution to this task would have far-reaching ramifications in robotics,
Figure 3.11: Average frame segmentation timing breakdown for VGA and QQVGA resolutions using the boundary mask and 50% downsampling of node potentials. Feature computation dominates and the graph cuts solver does not preclude real-time operation even at VGA resolution; thus, real-time segmentation at higher resolutions is conceivable with further optimizations to the feature set. Best viewed in color.
Figure 3.12: Example data labeling pipeline demonstrating one possible application of the STRGEN algorithm. Segmented objects are obtained while the raw data is being recorded. Segmentation hints from the user generate node potentials in the CRF so the user can direct the segmentation at any time; without input, the segmenter makes use of only the other cues discussed in Section 3.3.3 to produce segmentation masks. The output is a set of segmented object examples suitable for training standard object detectors.

computer vision, and graphics, opening the door for easier-to-train and more reliable object recognition, model capture, and tracking-based semi-supervised learning. While there remains more work to be done before a completely robust solution is available for unassisted segmentation and tracking, we believe this approach to be a promising direction for future research.
Figure 3.13: Object detection example trained using the method in Figure 3.12. Color and depth templates are extracted from segmented training objects; an appropriate object detection method (LINE-MOD in our example) searches for the templates in previously-unseen test frames and identifies “detections” as matches with responses exceeding a threshold.
Chapter 4

Conclusions

4.1 Summary

- STAC is a new approach to object recognition at the same level of abstraction as sliding windows or semantic segmentation. It requires model-free segmentation and tracking but magnifies user annotation efficiency (because entire tracks rather than individual views are the unit of annotation), improves classification accuracy (because classifications are aggregated over different views of an object), and enables group induction.

- Group induction is a new form of semi-supervised learning which exploits group structure in unlabeled data to automatically harvest new and useful training examples. It has been shown experimentally to reduce the need for user annotations by an order of magnitude without significantly reducing final accuracy.

- In environments where model-free segmentation and tracking is not freely available, the STRGEN framework is a good starting point for propagating object segmentations with minimal simplifying assumptions. It has the potential make STAC and group induction more broadly applicable, and could be useful in its own right for user-guided object selection or model capture.
4.2 Pipeline learning

For both the classification and segmentation frameworks presented in this dissertation, a highly multi-descriptor approach has proven essential. In object recognition, this was seen in the use of speed, size, spin images, histograms of oriented gradients, and more in the frame classifier. In segmentation, this was seen in the use of many different cues like optical flow, a local appearance model, bilateral filters, iterative closest point model alignments, depth changes, color changes, surface normal changes, and more. Generally, given sufficient training data, richer models produce more accurate results. For classification in particular, group induction makes acquisition of new training data quite easy, so we should feel free to continue adding descriptors until generalization error is sufficiently low. Further, this trend towards multi-descriptor systems is likely to accelerate as new sensor modalities become available. Affordable heat sensing is one example on the horizon.

In the course of implementing a highly multi-descriptor approach to problems like track classification or STRGEN, there are often hundreds of small choices that one must make, ranging from the size of the local region used to compute surface normals to the number of differently-parameterized spin images to use. The kind of choices that I highlight here are those which do not readily admit optimization via the usual methods like boosting, logistic regression, or structural SVMs. Instead, they tend to be chosen by “eye” or by cross-validation. As practical systems use more and more descriptors, graduate student descent will become an increasingly non-viable method of making these many small choices.

Instead, we need a method of automatically making these choices at training time. The machine learning expert should provide the general intuition, such as “speed could be useful for distinguishing bicyclists and pedestrians at long range” and “virtual camera views should make identifying cars easier,” but the machine should automatically determine details such as the smoothing parameter for speed estimation and whether all three of the top-down, long-side, and short-side views are necessary. In particular, this is not just about choosing parameters that are hard to numerically optimize; it is also about making decisions that affect the number of
parameters, such as how many different variations on a descriptor space to employ.

Moreover, in real-world systems that must run in real time, the caching of intermediate results is essential. For example, in Chapter 1, three different virtual camera images are generated from the same cached canonically-oriented point cloud, and several different HOG descriptors are computed from the same cached virtual camera image. This structure is just one part of the directed acyclic graph of computation that produces the complete set of object descriptors: the descriptor pipeline. This structure provides both caching of intermediate results and the opportunity for parallel execution of compute nodes.

This motivates the automatic learning of full descriptor pipelines. Ideally, this method would A) determine the details of a descriptor pipeline that are time consuming to choose manually, B) implicitly reason about the effects of caching intermediate results on runtime efficiency, and C) understand the need for statistically significant results when the evaluation is stochastic. One way to approach this problem is to think of it as cross-validation on steroids: try many different descriptor pipeline structures, train a model on a training set, make predictions on a validation set, repeat until a sufficiently good model is found, then do a final evaluation on an unseen test set. A somewhat more sophisticated approach would be to model this procedure as something akin to a multi-arm bandit problem, where there are a number of pre-defined transformations that can be made to the descriptor pipeline (i.e. the “moves”) along with a hidden and changing reward distribution associated with each possible move.

There is a connection here to deep learning, which attempts to learn good descriptor spaces from completely unlabeled data. In practice, however, one typically cares about an end application, and one must make many small structural choices about the deep learning architecture such as how many levels to use in the hierarchy, how they are connected, pooling size, local contrast normalization size, and so on. Automatic optimization of this structure is essentially another instantiation of pipeline learning.
CHAPTER 4. CONCLUSIONS

4.3 Variations on group induction

Combination with the ADBF. The augmented discrete Bayes filter described in Chapter 1 leads to significant improvement in track classification accuracy, but its need for a held-out training set significantly complicates matters for group induction, leading to the decision to use a more standard discrete Bayes filter for runtime track classifications in Chapter 2. It is unclear how to best combine these two mathematical frameworks. Fitting an ADBF as a post-processing step after group induction training is an easy and obvious approach, but then group induction does not benefit from the improved track classifications produced by the ADBF. It is possible that some deeper mathematical motivation besides alternating optimization is required to enable a principled combination of the ADBF and group induction.

Dynamic group induction. Group induction as presented in Chapter 2 is designed for working with class labels that do not change throughout the course of a track. This is reasonable for many object classes, but one can imagine scenarios in which this is not the case. For example, it might be desirable for an autonomous car to recognize a pedestrian hailing a cab or a bicyclist signaling left. These can be encoded as different object classes, but now the class label can be expected to change mid-track. For these kinds of problems, group induction can be reformulated as an optimization problem in which the class label for every frame is a different optimization variable, and in which the objective contains terms expressing a penalty for changing the inducted label mid-track. A reasonable name for this optimization is dynamic group induction because it is related to group induction, assumes the use of temporal data, and admits a dynamic programming solution.

Other sources of group structure. Group structure is not exclusive to the STAC approach. For example, consider the problem of moving a depth sensor through a static 3D environment, then producing a pointwise segmentation of the 3D model of that environment into road, grass, building, foliage, and ‘other’ classes. Group structure exists here, too, because a small region in space has a single hidden label and many individual views. Intuitively, if a particular small region looks like foliage from most views, then the remaining views can likely be added to the training set.
as new and useful examples of foliage. More practically, this idea could be used for training a vision system to recognize a particular set of crop plants, weeds, and dirt in a farming context.

**Combination with co-training.** Finally, combining the intuition of group induction with that of co-training [7] is appealing. Despite the multi-modal nature of many robotic perception problems, co-training-like approaches are rare in the literature.
Bibliography


[64] Bo Wu and Ram Nevatia. Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors. *International Journal*
