# CANSS: A Candidate Selection and Search Algorithm to Initialize Car Tracking

Frank Dellaert

October 1997

CMU-RI-TR-97-34

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

## Abstract

This report describes an algorithm to find cars in images taken by a forward-looking camera, mounted on a pursuing vehicle. It was developed in order to initialize a car tracker described elsewhere [2]. The algorithm proceeds in two steps, in order to find a bounding box that most probably corresponds to a car to be tracked. In a *candidate selection* step, potential edges for each of the 4 sides of the bounding box are selected. This is done using non-parametric density estimation followed by detection of local maxima. Then, in the *search step*, the most probable bounding box is selected from a set of hypotheses obtained by combining the candidate edges in all possible ways. The resulting algorithm is simple, fast, and works well in practice.

This work was supported in part by USDOT under Cooperative Agreement Number DTFH61-94-X-00001 as part of the National Automated Highway System Consortium. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of the United States Government or any of the sponsoring institutions.

## 1 Introduction

This report presents an algorithm for car detection that will be used to initialize a car tracking system. This work is done in the context of the Automated Highway System project (AHS), and our primary goal is to provide autonomous vehicles with spatial awareness. One important component to that is the ability to detect and track other vehicles on the road. To that aim, we have developed a model-based vision system [2] that tracks vehicles in front of the autonomous vehicle, by processing the images from a forward looking camera mounted under the rear-view mirror. However, this tracking system only works when a good initial estimate of the position of the tracked car is available.

Thus, before we can start tracking a given car, we need to detect it in the first frame, without having the benefit of an initial guess as to its position. The remainder of this paper presents an algorithm that does exactly this, with adequate performance and while satisfying the real time constraints. Although detection of an object in an image can in general be a difficult problem, the AHS domain imposes some constraints that we can use to our advantage. (1) If we restrict our attention to cars in front of the ego-vehicle, we can use the fact that cars appear as approximately square boxes when seen from behind. (2) Cars do not appear in random places in the image: the geometry of the environment makes that the size and position of cars in the images follow a typical distribution. (3) In the video data we work with, cars are dark with respect to the road and the background sky.

We will attempt to detect cars by searching for *bounding boxes* in the image that most probably correspond to cars on the road. For any given rectangle within the image boundaries we can look at the image properties associated with the rectangle to decide whether it is likely to correspond to a bounding box around a car. We do this by deriving a fitness measure which is a function of the rectangle geometry and the image 'under' the rectangle. Intuitively, this fitness function will measure how strong the edges are along the rectangular contour, and how likely it is that a given rectangle geometry corresponds to a car a priori. Finally, we simply pick the rectangle with the highest fitness value as the one which most likely corresponds to the bounding box around a car.

To make this search for the best bounding box tractable, we decompose the problem into two manageable parts: (1) selection, and (2) search. In the *candidate selection* step, we look for candidate rows and columns that are likely to contain one edge of a car generated bounding box. In the *search* step, we search over combinations of these candidates to optimize the fitness function as discussed above. This two step process constitutes the basis for the technique, which we have called the *Candidate Selection and Search* algorithm.

The *candidate step* is now described. Even if we restrict our attention to finding bounding boxes around cars, there are still too many rectangles in any given image to search over by brute force. To make the problem tractable, we rely on the following observation: if there is a car in the image, it must have a top, a bottom, a left and a right edge. Thus, we could simply search for the few columns and rows that are likely to contain one of these bounding box edges, since for our domain the bounding boxes around cars will almost always be aligned with the horizontal and vertical image axes. Unless the image is littered with strong horizontal and/or vertical edges, there is a good chance that these bounding box components are the only strong edges, or one of a few, at that particular height or horizontal position in the image. That allows us to select these candidate rows and columns by simply looking at some global properties of each column/row, e.g. the sum of the gradient over an entire column. To make optimal use of these calculated quantities, we will employ pattern recognition techniques, as described below.

Next, a *search step* searches over all candidate rectangles. Once we find a given number of candidate rows and columns for each constituent part of the bounding box, we can combine them into rectangles to be searched over. However, the number of rectangles to be search is now considerably reduced, and a brute force examination of all rectangles is manageable within the real time constraints.

In summary, we follow the following procedure: for every image we want to find cars in, we

1. Calculate a set of edge-related features along every row and column in the image
2. Find the rows and columns where a car bounding box edge might be present
3. Combine those candidates rows and columns into a candidate set of rectangles
4. Search the set for the rectangle with the highest fitness value

The remainder of the report is structured in four sections. In Section 2 we discuss the reasoning behind and the algorithms used for finding candidate edges. Section 3 will detail how the search over the combination rectangles is done, specifically what fitness function we are optimizing and how we arrive at it. Results of the overall algorithm are discussed in Section 4. The last section (Section 5) concludes and discusses some open issues.

## 2 Candidate Selection

In the selection step, we use pattern recognition techniques to rank and select candidate rows and columns that will be used in the search step. In particular, we want to pick out the rows and columns that are likely to contain either a top, bottom, left or right edge of a bounding box around a car. Since our assumption is that such a bounding box will have strong edges on all sides, one approach is to simply look at the average gradient along a row (or column) and select those with the highest value. However, that approach does not take into account that (a) rows at a certain height in the image are more likely to contain say, the top edge of a car, just by virtue of their position in the image, and (b) maybe other properties different from the average gradient are more strongly correlated with what we are after. For example, columns at the very right of the images have almost zero probability of containing a left edge. In addition, for all we know the *maximum* gradient along a column is more indicative of a car edge, rather than the *average* gradient. Rather than try to take into account all these factors in an ad-hoc fashion, our approach is to calculate a number of properties for each row and column (including position itself), and work with standard pattern recognition techniques to calculate a posterior probability for each column/row. The local maxima thereof can then be singled out as the candidates to be passed on to the search step.

In particular, if we restrict our attention temporarily to the left edge of the car, for each column in the image we will calculate $P(L|\mathbf{x})$, the posterior probability of the column containing a left edge (the event $L$) given some feature vector with data about the column (the real-valued vector $\mathbf{x}$). We can use the Bayesian probability framework to answer this question in a principled manner. Via Bayes law we can express the posterior probability $P(L|\mathbf{x})$ as:

$$P(L|\mathbf{x}) = \frac{P(\mathbf{x}|L)P(L)}{P(\mathbf{x})}$$

where $P(\mathbf{x}|L)$ is the likelihood of $L$ given $\mathbf{x}$, and $P(L)$ is its prior probability. $P(\mathbf{x})$ is a normalization factor to make $P(L|\mathbf{x})$ a probability. We will estimate $P(\mathbf{x}|L)$, $P(L)$, and $P(\mathbf{x})$ from training data, and then apply Bayes law to arrive at $P(L|\mathbf{x})$.

## 2.1 Training data

Since the intended application of this algorithm is to initialize car tracking in the AHS domain, the images we use for training and testing the algorithm are taken from this domain. One of our AHS research vehicles, the Navlab 8 -an Oldsmobile Silhouette mini-van- was used to obtain a 25 minute segment of video recorded from a forward looking camera mounted below the rear-view mirror of the Navlab. During this time, the Navlab was being driven manually on a stretch of I-79N outside Pittsburgh, except for a short time where the driver went off and on the highway to return to base. The camera was mounted on a height of 1.42 m, measured from the road surface, and tilted downwards by about 10 degrees. The images were digitized using the built-in hardware of a Silicon Graphics O2 workstation, and the resulting digitized frame size is 640 pixels wide by 480 pixels high. However, in what follows we work with fields instead of frames, since field interleaving gives rise to strong horizontal edges across the image, due to the time lag between two fields and the distance the car moves in that interval. In addition, we drop all even columns, to retain the aspect ratio of the original image and enabling us to use uniform blurring filters as part of the edge detection operation, while saving us some computation as well. Thus, the final size of the images we work with is 320 by 240 pixels.

From this video segment, 100 images were selected at random to generate training data, and an additional 100 were selected for testing purposes. For each of these 200 images, we manually labeled all the cars in the image that were completely visible. In Figure 1 we have shown the outlines of all the labeled cars in the training set, superimposed in

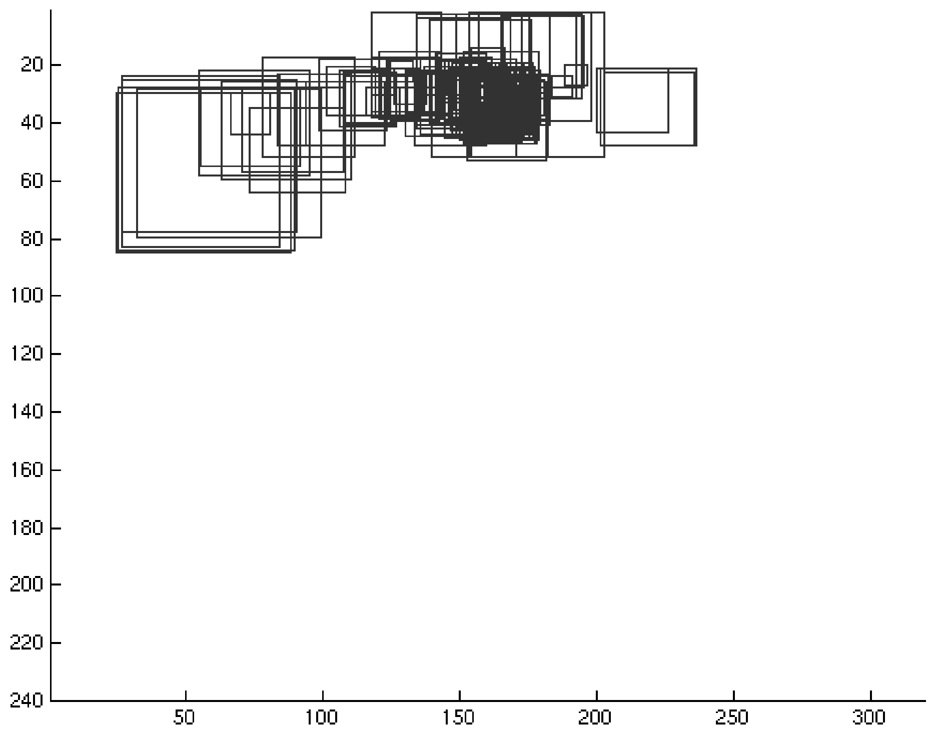

Figure 1: Distribution of labeled cars in the training set

one image. As you can see, there is a non-uniform distribution of car positions and sizes within the image, a fact which we will take advantage of several times. In particular, we limit our attention to a sub-window of size 100 by 200 pixels that contains most of the labeled cars in the training set, since there is no use in processing large parts of the image where we know a car is unlikely to appear. This also removes potentially distracting features from the image, strengthening our assumption that a car edge will be one of few strong edges along a given row or column.

For each windowed image in the training and test set we calculate 300 feature vectors $\mathbf{x}$, one for each of the 100 rows and one for each of the 200 columns. In each case, $\mathbf{x}$ is a real valued 7-dimensional vector $[p, min, avg, max, min_d, avg_d, max_d]^T$, where:

- $p$: the position of the row or column (column index $u$ or row index $v$)
- $min$: the minimum edge value over the row/column
- $avg$: the average edge value
- $max$: the maximum edge value
- $min_d$: the derivative of $min$ with respect to $p$, for a given image

- $avg_d$: the derivative of $avg$
- $max_d$: the derivative of $max$

These quantities are obtained by calculating a horizontal and vertical gradient image for each windowed image, using a conventional Sobel operator in each case, blurred with a Gaussian of $\sigma$=1.5. These edge images are then processed row by row (and column by column) to obtain the $min$, $avg$, and $max$ values. These values are then differentiated with respect to $p$, the position of the row (column), to obtain $min_d$, $avg_d$and $max_d$. Note that the position of the row or column in the image is itself one of the features.

Because we have 200 images, each generating 300 training or test samples, we end up with a total of four datasets:

- 10000 row samples ($\mathbf{x}$,$T$,$D$) for training
- 10000 row samples ($\mathbf{x}$,$T$,$D$) for test
- 20000 column samples ($\mathbf{x}$,$L$,$R$) for training
- 20000 column samples ($\mathbf{x}$,$L$,$R$) for test

where $T$, $D$, $L$, and $R$are the classification labels corresponding respectively to the top, bottom, left, and right edge of the cars.

## 2.2 Estimating $P(L|\mathbf{x})$using Kernel Regression

We have used *kernel regression*, a Memory Based Learning technique (abbreviated MBL, see e.g. [1]), to estimate the posterior probability of each of the four classifications from the training data. Kernel regression (KR), or locally weighted averaging, follows a very simple algorithm. If we again restrict our attention to the left edge $L$ case, KR proceeds as follows:

- Collect a large number of training data, each sample a $(\mathbf{x}, L(\mathbf{x}))$ tuple (as described above)
- Construct a new dataset $(\mathbf{x}, f(\mathbf{x}))$, where $f(\mathbf{x}) = 1$ if $L(\mathbf{x}) = TRUE$, and 0 otherwise.
- Now estimate the value of $P(L|\mathbf{x})$by forming a weighted average:

$$P(L|\mathbf{x}) \approx \frac{\sum_{i=1}^{N} G(\mathbf{x}; \mathbf{x}_i, \sigma) f(i)}{\sum_{i=1}^{N} G(\mathbf{x}; \mathbf{x}_i, \sigma)} \tag{1}$$

  where $G(u; \mu, \sigma)$ is a Gaussian weighting function, and the standard deviation $\sigma$ is a parameter.

## 2.3 Deriving Kernel Regression

Below we show the validity of using kernel regression (equation 1) to estimate a posterior probability. Remember that we want to estimate the posterior probability $P(L|\mathbf{x})$, given by

$$P(L|\mathbf{x}) = \frac{P(\mathbf{x}|L)P(L)}{P(\mathbf{x})} \tag{2}$$

Using Parzen window density estimation [3] we can estimate densities placing a Gaussian kernel on each of the data points in a given set, and approximating the density of the set by a sum of all these Gaussians (appropriately normalized). In particular, if $N_L$ and $N_{\bar{L}}$ are the number of samples where $L$ is TRUE and FALSE, respectively, then we can estimate the conditional densities $P(\mathbf{x}|L)$ and $P(\mathbf{x}|\bar{L})$ by:

$$P(\mathbf{x}|L) \approx \frac{1}{N_L}\sum_{i\in L} G(\mathbf{x};\mathbf{x}_i,\sigma) \tag{3}$$

$$P(\mathbf{x}|\bar{L}) \approx \frac{1}{N_{\bar{L}}}\sum_{j\in \bar{L}} G(\mathbf{x};\mathbf{x}_j,\sigma) \tag{4}$$

where $G(u;\mu,\sigma)$ is the multivariate Gaussian kernel, and $\sigma$ acts as a smoothing parameter, to be chosen judiciously as a function of the smoothness of the underlying density and the amount of training data available.

The prior probabilities $P(L)$ and $P(\bar{L})$ can be estimated simply as:

$$P(L) \approx \frac{N_L}{N} \tag{5}$$

$$P(\bar{L}) \approx \frac{N_{\bar{L}}}{N} \tag{6}$$

We can now can plug the above formulas into Bayes law (Equation 2) to arrive at:

$$P(L|\mathbf{x}) \approx \frac{\frac{1}{N_L}\sum_{i\in L} G(\mathbf{x};\mathbf{x}_i,\sigma)\frac{N_L}{N}}{\frac{1}{N_L}\sum_{i\in L} G(\mathbf{x};\mathbf{x}_i,\sigma)\frac{N_L}{N} + \frac{1}{N_{\bar{L}}}\sum_{j\in \bar{L}} G(\mathbf{x};\mathbf{x}_j,\sigma)\frac{N_{\bar{L}}}{N}} \tag{7}$$

$$= \frac{\sum_{i\in L} G(\mathbf{x};\mathbf{x}_i,\sigma)}{\sum_{i\in L} G(\mathbf{x};\mathbf{x}_i,\sigma) + \sum_{j\in \bar{L}} G(\mathbf{x};\mathbf{x}_j,\sigma)} \tag{8}$$

By assigning a real value $f(i) = 1$ to the samples in the set $L$, and $f(i) = 0$ to those in $\bar{L}$, we obtain:

$$P(L|\mathbf{x}) \approx \frac{\sum_{i=1}^{N} G(\mathbf{x};\mathbf{x}_i,\sigma)f(i)}{\sum_{i=1}^{N} G(\mathbf{x};\mathbf{x}_i,\sigma)} \tag{9}$$

## 2.4 Feature Selection and Smoothing

Several questions still remain: (a) what should we use as the *kernel width*, i.e. standard deviation $\sigma$ of the kernel, and (b) should we weight all features equally ? Especially the latter point is interesting: it is obvious that there will be a lot of redundancy in the feature vector as described above. For example, minimum and maximum edge value are expected to be strongly correlated in an edge rich region.

The answers to these questions can be found by searching over a range of possible kernel widths and combinations of features. Leave-one-out cross-validation was used to rank the models according to their mean-square-error performance on the dataset. We used a hold-out set of 1000 samples to prevent over-fitting on the test-set. The best model found used three features, equally weighted:

1. position $p$ (the strongest predictor of class)
2. average edge value $avg$ (a fair predictor)
3. the derivative of the average $avg_d$ (a fair predictor)

The rest of the features remained unused, indicating they were redundant for the purposes of classifying the columns and rows according to the edge they contain (or not).

## 2.5 Candidate Selection

Finally, candidates for the top, bottom, left, and right edges of potential bounding boxes are then obtained by extracting the local maxima of the respective posterior probability estimates $P(T|\mathbf{x})$, $P(D|\mathbf{x})$, $P(L|\mathbf{x})$, and $P(R|\mathbf{x})$. Typically 3 to 5 candidates are selected for each edge.

# 3 Searching over Candidate Boxes

Once we obtain a number of candidates, we can combine them to form hypotheses over 'box space'. We then simply pick the most probable box from the possible candidate boxes. Since the number of possible box hypotheses is greatly reduced by the edge candidate step, we can afford the computation to consider each box in turn and determine its place in the ranking. Again we can use Bayes law to compute the posteriori probability for each box $\mathbf{u}$ given the image data $\mathbf{z}$:

$$P(\mathbf{u}|\mathbf{z}) = P(\mathbf{z}|\mathbf{u})P(\mathbf{u})/P(\mathbf{z})$$

Here $\mathbf{u}$ is a 4-dimensional vector, denoting the position in the image $(u, v)$ and the width and the height $(h, w)$ of a hypothetical bounding box, and $\mathbf{z}$ will be discussed shortly.

For the prior probability we assume a normal density

$$P(\mathbf{u}) = N(\mu, \Sigma)$$

where$\mu$ and $\Sigma$ are the maximum likelihood estimates of the mean and covariance matrix, respectively. They are estimated from the training data.

The likelihood $P(\mathbf{z}|box = \mathbf{u})$ is harder to obtain, since it is clearly impossible to enumerate all possible images, let alone integrate over them to normalize any measure to a valid probability density. Instead we will work by defining an energy measure that we will seek to minimize, and assuming the probability is given via the Boltzmann distribution as:

$$P(\mathbf{z}|\mathbf{u}) = \frac{1}{Z} exp(-E_{likelihood}(\mathbf{z}, \mathbf{u}))$$

Here Z is the normalization factor that we will not be able to compute. However, since we are only interested in the ranking of probabilities, we only need to rank the energy term $E_{likelihood}(\mathbf{z}, \mathbf{u})$. Thus, we need to minimize the following quantity:

$$E_{posterior}(\mathbf{z}, \mathbf{u}) = \alpha E_{likelihood}(\mathbf{z}, \mathbf{u}) + E_{prior}(\mathbf{u}) \tag{10}$$

where $\alpha$ is a factor that balances the prior against the likelihood. The factor $\alpha$ is determined by trial and error, and will be discussed in more detail below.

The likelihood energy $E_{likelihood}$ we use is the the average edge value along the contour defined by $\mathbf{u}$, i.e. the same energy term used in [2] for purposes of tracking cars.

Since the form of the prior was assumed Gaussian, $E_{prior}$ is easily seen to be:

$$E_{prior} = \frac{1}{2}(\mathbf{u} - \mu)^T \Sigma^{-1} (\mathbf{u} - \mu) + k$$

where k is a constant that can be dropped for purposes of ranking or minimization.

## 4 Results

A typical, qualitative picture of how the algorithm performs is shown in Figure 2. In the figure, a video-still is shown with 2 cars in it. For each side of the potential bounding box, three candidate edges are selected, shown as dark horizontal and vertical lines, with either red or green emphasis where strong edges are detected. For horizontal lines, green edges correspond to the top, red edges to the bottom of the bounding box. For vertical edges, green corresponds to left, and red to right. Of the 81 generated bounding boxes, the maximum probable one (i.e. minimizing the sum of $E_{prior}$ and $E_{likelihood}$ in equation 10) is shown in yellow.

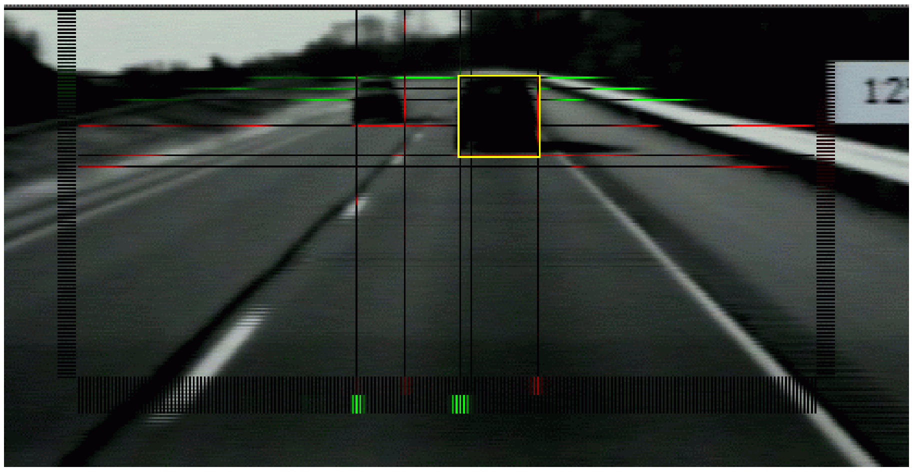

Figure 2: Candidate edges and most probable box (in yellow) for a typical input image.

| $\alpha$ | correct | percent |
|---|---|---|
| 0 | 36 | 38.30 |
| 0.125 | 63 | 67.02 |
| 0.25 | 64 | 68.09 |
| 0.5 | 60 | 63.83 |
| 1 | 57 | 60.64 |
| 2 | 51 | 54.26 |
| 4 | 41 | 51.06 |

Table 1: Percentage of correctly classified cars for different values of $\alpha$, using 3 candidates per edge (generating $3^4 = 81$ box hypotheses).

In order to provide a more quantitative assessment of performance, we measured the percentage of cars correctly classified in the test-set of 100 images. Since 6 of the test images contained no car, they were excluded from the set, for a total of 94 test images. Table 1 shows the results for different values for $\alpha$. As can be seen by looking at the first row, with $\alpha = 0$, by simply ignoring the image we can identify almost 40 % of the cars. That seems impossible until you realize that the image has of course been used to generate the set of hypothesis boxes in the first place. The prior simply picks out the most a priori likely box from those candidates. The fact that it does so well indicates that our prior is quite good (i. e. the Gaussian assumption must not have been too far from the true distribution), and that the candidate step did well. As $\alpha$ goes up, we quickly reach a maximum but then performance drops off fast. This indicates that by looking at the average edge value alone we will choose the wrong box on many occasions.

| $\alpha$ | correct | percent |
|---|---|---|
| 0 | 24 | 25.53 |
| 0.125 | 66 | 70.21 |
| 0.25 | 63 | 67.02 |
| 0.5 | 47 | 50.00 |
| 1 | 36 | 38.29 |
| 2 | 28 | 29.78 |
| 4 | 26 | 27.65 |

Table 2: Percentage of correctly classified cars for different values of $\alpha$, using 5 candidates per edge (generating $5^4 = 625$ box hypotheses).

The other free parameter is the number of candidate edges that we extract for each side of the bounding box. Table 1 was generated with this number equal to 3. It was chosen so low because we get quickly punished by choosing a high value for this parameter, as the number of hypothesis boxes goes up by the power of 4. However, it might be advantageous to use a higher value, since an actual car edge might be not be so clear and might be ranked lower on the list than false positives generated by other image features. The more candidates we admit, the less likely that will happen. Table 2 was generated with the number of extracted edges set to $5$, and we can see that the performance does indeed increase, though not by a significant amount. Note also that a shift occurs towards trusting the prior even more, indicating that the new candidate edges created some confusion that needed to be sorted out by the prior.

In summary, we obtain correct recognition in about 70% of the cases, which is good enough for the purpose of initializing tracking.

## 5 Discussion and Open Issues

We reached our goal of detecting cars in images, and performance is adequate for purposes of initializing car tracking. We presented a two-tiered approach to detecting cars in video-images with no prior estimate: (a) find a number of plausible candidate lines that contain top, bottom, left or right edge of a car, and (b) search the most likely bounding box over the reduced space of such boxes formed by combining them. Performance is good: up to 70% of all cars in the training set are correctly localized.

In order to further improve performance, we could look at the average grayscale within the bounding box as an additional clue. Another idea is using edges formed by differencing two successive frames, which would exclude static edges on the cars themselves from consideration. Also, it might be beneficial to model the existence of light and dark cars, and treat them separately. In the particular lighting conditions on the one tape we used throughout, cars were consistently dark. However, under different conditions we might see many more cars that appear light on the background.

A number of issues remain to be addressed in order to make the technique more useful in a real-world application. One obvious shortcoming is the inability of the algorithm to deal with partially visible cars and occlusion. Also, we should have a better handle on when even the most probable box is probably not a car, i. e. we should have some rejection thresholds.

However, preliminary experiments in combination with tracking, suggest that the technique works well and is fast enough to initialize tracking in a robust fashion. Thus, although there are possible improvements, the algorithm we presented is simple, fast, and works well in practice.

## References

[1] C. G. Atkeson, S. A. Schaal, and A. W. Moore. Locally weighted learning. *AI Review*, 11:11–73, 1997.

[2] Frank Dellaert and Chuck Thorpe. Robust car tracking using Kalman filtering and Bayesian templates. In *Proc. of the SPIE - Int. Soc. Opt. Eng.*, volume 3207, October 1997.

[3] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York, 1973.