

Pattern Recognition Techniques to Infer Driver Intentions

Hiren Mansukhlal Mandalia, CS, Drexel University

Technical Report DU-CS-04-08
Department of Computer Science
Drexel University
Philadelphia, PA 19104
December 2004

Pattern Recognition Techniques to Infer Driver Intentions

A Thesis

Submitted to the Faculty

of

Drexel University

by

Hiren Mansukhlal Mandalia

in partial fulfillment of the

requirements for the degree

of

Master of Science in Computer Science

December 2004

Acknowledgements

I take this opportunity to thank Dr. Salvucci, Dr. Shokoufandeh and Dr. Lee for their support and guidance throughout the writing of this thesis. I would also like to thank Nobuyuki Kuge and Tomohiro Yamamura at Nissan Motor Co., Ltd., Japan for collecting the driving data. I'm also grateful to my family, friends and colleagues who have been a constant source of inspiration to me throughout this study.

Table of Contents

List of Tables	vi
List of Figures.....	vii
ABSTRACT.....	ix
1. Introduction.....	1
1.1 Recent Work	2
1.2 Proposed Approach.....	4
1.2.1 Defining a Lane Change	5
1.2.2 Continuous vs. Discrete Recognition.....	6
1.2.3 Data Collection	7
2. Support Vector Machine Recognition	9
2.1 Brief description of Support Vector Machines	9
2.2 Motivation for SVM	11
2.3 Kernel Selection and Data Representations.....	12
2.4 Training Support Vector Machines for LC/LK	13
2.4.1 Choosing a Window of Time.....	13
2.4.2 Choosing a Data Representation.....	16
2.4.3 Choosing a Feature-set.....	19
2.4.4 Generating Continuous Recognition.....	20
2.5 Results.....	20
2.6 Software Package for SVM	23
3. Hidden Markov Model Recognition.....	24
3.1 Brief Introduction to HMM	25

3.2 Motivation.....	26
3.3 Developing HMM-based Recognition System.....	26
3.3.1 Choosing the HMM Structure.....	26
3.3.2 Analogy from Speech Recognition.....	27
3.3.3 Training HMM for LC and LK.....	28
3.3.4 Choosing the Feature Set.....	30
3.4 Recognition.....	31
3.5 Results.....	32
3.6 HTK - Development tool.....	34
4. Results and Comparative Study.....	35
4.1 ROC curves.....	36
4.2 Time Elapsed.....	37
4.3 Lateral Movement.....	38
5. Graph theory based Lane Change Detection.....	42
5.1 Introduction.....	42
5.2 Overview of the Technique.....	43
5.2.1 From Continuous Stream of Data to Correlation Matrices.....	43
5.2.2 From Correlation Matrices to Graphs.....	44
5.2.3 Construction of the Database.....	45
5.2.4 Simple Graph-Matching Algorithm.....	47
5.3 Results.....	50
6. General Discussion.....	52
List of References.....	55

APPENDIX A..... 58

List of Tables

Table 1: SVM Feature Set.....	22
Table 2: Percentage Detected at 1 and 5 % False Positives (non-Overlapping).....	22
Table 3: Percentage Detected at 1 and 5 % False Positives (Overlapping).....	22
Table 4: Final Results with Different Kernels	23
Table 5: Word Dictionary and Structures	28
Table 6: HMM Feature Set	33
Table 7: Percentage Detected at 1 and 5 % False Positives (HMM Structure I).....	33
Table 8: Percentage Detected at 1 and 5 % False Positives (HMM Structure II).....	33
Table 9: Results with Graph Technique.....	51
Table 10: Complete Feature Set.....	58

List of Figures

Figure 1: Example to Show Different Driver Actions	1
Figure 2: Lane Change Definition	5
Figure 3: Discrete vs. Continuous Recognition	6
Figure 4: Screen Dump of the R2 Video	8
Figure 5: Points Separated by Hyperplane.....	10
Figure 6: Steering Angle Displays a Sine-wave like Pattern.....	14
Figure 7: Moving Window of Constant Size	15
Figure 8: Data Representation	18
Figure 9: A Left-Right HMM	27
Figure 10: Configuration of ‘LCL’ Word (using Structure II)	29
Figure 11: Training Windows for LC/LK.....	29
Figure 12: Input Test Sentence	31
Figure 13: Mind-Tracker Overview	35
Figure 14: ROC Curves for the 3 Frameworks.....	36
Figure 15: Time Elapsed Chart.....	37
Figure 16: Lateral Movement Chart	39
Figure 17: ROC Comparison Chart	40
Figure 18: Time Elapsed Chart.....	41
Figure 19: Lateral Movement Chart	41
Figure 20: Continuous Stream to Matrices	44
Figure 21: Matrix to Graphs	45
Figure 22: Construction of Database	47

Figure 23: Matching and Data Mining Strategies.....	48
Figure 24: Degree of Matching.....	49

ABSTRACT

Pattern Recognition Techniques to Infer Driver Intentions

Hiren M. Mandalia

Dr. Dario Salvucci, Ph.D.

Driving is a complex task that requires constant attention from the mind and the body. Automobile drivers today are under high risks, thanks to the ever-expanding telematics industry, cell-phone driving and other distractions. Inferring driver intentions, especially critical ones like changing lanes, is therefore necessary for any intelligent driver support system. This thesis explores different methods to infer driver's intention to change lanes. Experimental data were collected that observed driver's behavior (e.g. speed, steer angle, gas pedal pressure) and environmental data around the driver (e.g. distance of the car in front). With the hypothesis that such data would display significantly different patterns during a lane change versus lane keeping, this problem was formulated as a pattern recognition problem. Two different techniques were studied in detail to solve this problem, namely support vector machines (SVMs) and hidden markov models (HMMs). These two machine learning techniques showed promising results. SVMs have been particularly effective in early detection of lane changes with a very high sample-by-sample prediction rate. In addition, this thesis compares these techniques with a new "mind tracking" approach [9, 10] and also proposes a new graph-based approach to detect lane changes.

1. Introduction

Driving is a complex skill and is getting increasingly complex with the growing popularity of in-car electronics, navigational devices, cell phones and other telematics devices. With this complexity comes the increased risk of driving. Automobile manufacturers have continually sought to devise innovative driver support systems that can reduce such risks and make the driving experience more pleasurable. Such support systems provide various kinds of help to the driver while driving and especially during critical maneuvers such as changing lanes, sharp turns, etc. However, for any such system to be effective - i.e. offer the right kind of help at the right time - it is imperative that we know what the driver is doing or trying to do.

One can argue as to what is the real necessity to infer driver intentions. The most critical reason is safety. Let us look at an example to demonstrate this. Figure 1 shows a driver in a black vehicle rapidly approaching a slow-moving vehicle in the front. At this point the driver has two options as shown in the figure.

- (1) To accelerate and steer into the other lane
- (2) To slow down and stay in the same lane

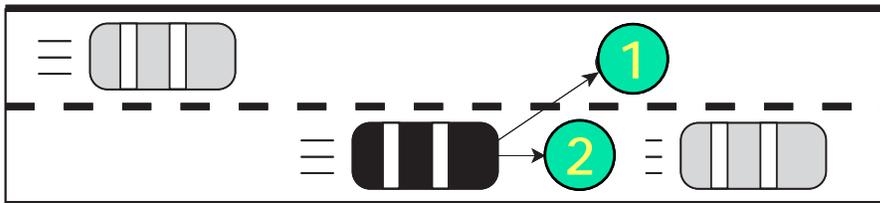


Figure 1: Example to Show Different Driver Actions

If the driver now decides to accelerate and steer into the other lane but the support system thinks that the driver is going straight and helps to slow down, an accident could be unavoidable. Let us say on the other hand, if the driver decides to stay in the lane but the system thinks otherwise and helps to steer – again, an accident might occur. Such disasters can be avoided only if the support system can accurately infer the driver’s intention. Thus for any intelligent driver support system it is very essential to know the expected behavior of the driver. The earlier these intentions are inferred, the more safely the driver can utilize the support system.

One of the most common and critical tasks during driving is changing lanes, therefore it becomes imperative that driver support systems predict drivers’ intentions to change lanes accurately. This thesis explores the problem of inferring driver intentions from a pattern recognition perspective. It presents a background of related work in this field and then discusses in detail two pattern recognition frameworks investigated to achieve real-world solutions.

1.1 Recent Work

Vast attention has been given to the driving task in general but much less attention has been directed to lane changing, despite its ubiquity in common driving environments — such as highway driving, which accounts for roughly 70% of vehicle miles on American roadways [23]. Existing work on lane-changing behavior in large part emphasizes the decision-making aspects of the task, particularly gap acceptance and the decision of when to change lanes [24, 25]. Other studies have addressed different aspects

of lane changing, from behavioral aspects such as typical durations [26] to practical development of lane-change collision warning systems [27]. Very little work has been done on recognizing driving maneuvers, especially critical ones like lane changing.

Some recent endeavors in this direction include work from Pentland and Liu (1999), Kuge et al. (2000), and Salvucci and Siedlecki (2003). The first two approaches were based on the concept that human behavior is made up of a sequence of internal ‘mental’ states that was not directly observable. These approaches used a technique called hidden markov models (used in Speech Recognition) that are probabilistic models powered by robust expectation-maximization methods. Pentland and Liu reported that their recognition system could predict changes in the first 0.5 second of the maneuver. However their recognizer offers only discrete recognition as opposed to continuous recognition (as reported in this thesis). Second, Kuge et al. reported results with a continuous recognition system; however, their system only uses steering-based features and has no knowledge of the surrounding environment, which clearly affects whether and when people make lane changes. This thesis also proposes a hidden markov model based recognition system but uses a wide range of features including environmental features like lane position.

Salvucci and Siedlecki proposed a mind tracking approach [9, 10] that uses a cognitive architecture called ACT-R to model driver’s behavior. The mind tracking system essentially isolates a temporal window of driver data and extracts its similarity to several virtual drivers that are created probabilistically using a cognitive model. However

it does not utilize the fact that the measurable components indicative of the driver state may often reside in some high dimensional feature space and data can be easily classified by finding a linear separating hyperplane. This thesis reports the use of Support Vector Machines that address this issue optimally.

1.2 Proposed Approach

This thesis concentrates on detecting lane change intentions using two separate machine learning frameworks, Support Vector Machines (SVMs) and Hidden Markov models (HMMs). Each framework has been employed independently to infer driver behavior. The data used for the experiments were collected from real instrumented vehicle that recorded different behaviors of the driver. The data are highly multimodal in nature. Correct implementation of each framework requires thorough understanding of the driver's behavior (lane changing vs. lane keeping). Each feature was examined closely to infer any specific patterns or trends displayed particularly during the time course of a lane change. Studies about patterns during a lane change [15] have revealed that drivers exhibit specific patterns for steering, acceleration and eye gazes. Careful investigations revealed that many features in addition to those mentioned above also contributed significantly to exhibit distinct trends. These observations clearly suggested that the driving data vary significantly during a lane change against normal driving (lane keeping). This served as the prime motivation to use pattern recognition techniques to recognize specific behaviors. Currently these techniques are used specifically for detecting lane changes, but they can be applied to a fuller range of driver intentions such as left and right turns, stops, parking, etc.

1.2.1 Defining a Lane Change

To detect a lane change (LC) it is critical we define it clearly. Precisely what time window within an entire lane change instance should be marked as lane change? Often drivers make initial unsuccessful attempts to cross over to the other lane. There are many subjective choices for the definition of a lane change. However the definition of lane change used in [9] was found to be most reasonable. Accordingly, *a lane change starts when the vehicle moves towards the other lane boundary and never returns i.e. the final shift of the driver towards the other lane. Any reversal cancels the lane change. The lane change ends when the vehicle crosses the lane boundary.*

We also assume that the lateral movement of the automobile towards the other lane should be greater than or equal to 3.5 m/s, meaning that the LC can take at most 10 seconds. (Note that average width of the road is approximately 3.5 m.)

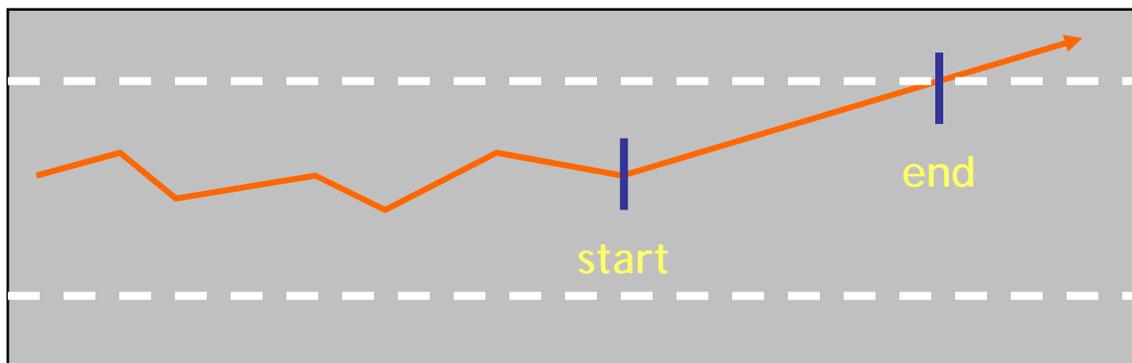


Figure 2: Lane Change Definition

Figure 2 shows the path followed by a driver during a typical lane change. Note that before making the final attempt to steer into the next lane, the driver made a few

unsuccessful attempts, which are ignored in the final definition. The time window defined by *start* and *end* marks the definition of lane change.

1.2.2 Continuous vs. Discrete Recognition

There are essentially two ways of framing the problem of lane change recognition: *continuous* (real-time) or *discrete* recognition. Discrete recognition essentially means recognizing driver intention after every discrete segment of time or when an event occurs. An event in this case would be a lane change (LC) or lane keeping (LK). Continuous recognition on the other hand produces recognition results continually. Driving data is a continuous stream of samples where each sample is a vector of feature values. Thus a continuous recognition will generate recognition output at every sample. Essentially, in discrete recognition entire block of samples is classified together. Figure 3 below diagrammatically describes this difference.

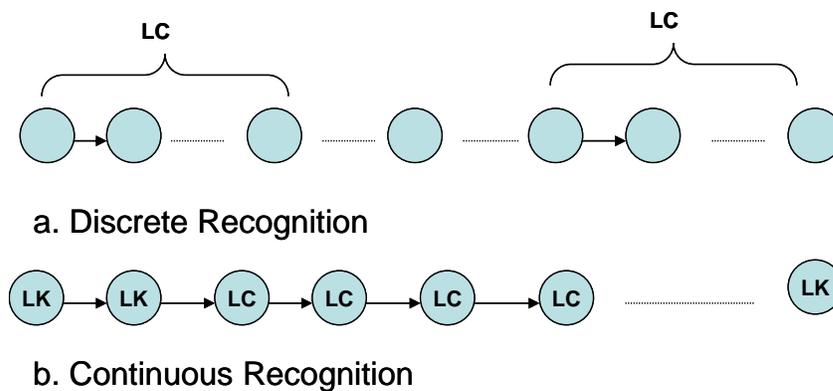


Figure 3: Discrete vs. Continuous Recognition

For an intelligent driver support system it is required that a continuous recognition output is generated. With the definition of *lane change* and *continuous recognition*, the problem domain is clearer. For a continuous pattern recognition system, every sample in the lane change window must be recognized as a lane change and others as lane keeping.

1.2.3 Data Collection

The data sets used for the experiments in this thesis were collected using a real instrumented vehicle at the Nissan Motor Company in Oppama, Japan. Four driver subjects were asked to drive on a Japanese multi-lane highway environment for one hour each. The drivers spent approximately one hour driving through smooth and dense traffic conditions. This provides us with a good representation of lane changes of different time lengths. The drivers were given no specific goals or instructions and they were allowed to drive on their own. Electronic sensors from the car captured values for driver actions like gas pedal pressure, steering angle, speed etc. Using digital cameras and advanced vision algorithms, environmental features around the driver were recorded. Some of these include distance of the lead car, lane position etc. Three additional cameras were installed inside the car that captured video images of (i) the driver (side profile), (ii) gas pedal and brakes movements, and (iii) front view (as viewed by the driver). However data from these video images were used only for initial analysis. Figure 4 below shows a screen dump of a video combining streams from different cameras. In each of the two machine learning frameworks discussed in the following sections 2/3rd of the data was used for training and the remaining 1/3rd was used for testing. A complete list of all the features with their brief explanation is provided in Table 10 in the Appendix A. The data was collected at a sampling rate of 10 Hz.



Figure 4: Screen Dump of the R2 Video

2. Support Vector Machine Recognition

This section discusses the use of Support Vector Machines (SVMs) towards continuous lane change detection. Support Vector Machines are learning machines that can perform binary classification (pattern recognition) and real valued function approximation (regression estimation) tasks. SVMs have been widely used for isolated handwritten digit recognition [1, 2], object recognition [3], speaker identification [4], and face detection in images [5] and text categorization [6]. The chapter reviews the basic functioning of SVMs, motivation for using SVMs for lane change detection, training and recognition of Lane Changes. The results, in terms of the prediction accuracy (true positive rates and false positive rates) and other measures are discussed in the concluding section.

2.1 Brief description of Support Vector Machines

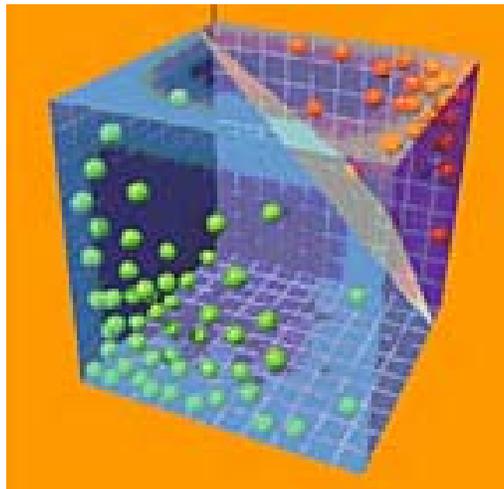
‘Support vector machines’ (SVM) are based on statistical learning theory that uses supervised learning [16]. In supervised learning, a machine is trained instead of programmed, to perform a given task on a number of input-output pairs. According to this paradigm, training means choosing a function which best describes the relation between the inputs and the outputs. The central question of statistical learning theory is how well the chosen function generalizes, or how well it estimates the output for previously unseen inputs.

In general, any learning problem in statistical learning theory will lead to a solution of the type

$$f(\mathbf{x}) = \sum_{i=1}^l c_i K(\mathbf{x}, \mathbf{x}_i)$$

where the \mathbf{x}_i , $i = 1, \dots, l$ are the input examples, K a certain symmetric positive definite function named kernel, and c_i a set of parameters to be determined from the examples. For details on the functioning of SVM readers are encouraged to refer to [1].

In short, the working of SVM can be described as statistical learning machines that map points of different categories from n -dimensional space into a higher dimensional space where the two categories are more separable. It tries to find an optimal hyperplane in that high dimensional space that best separates the two categories of points.



(Source: <http://www.support-vector.net>)

Figure 5: Points Separated by Hyperplane

Essentially, the hyperplane is learned by the points that are located closest to the hyperplane which are called support vectors. There can be more than one support vector on each side of the plane. Figure 5 shows an example of two categories of points separated by a hyperplane.

2.2 Motivation for SVM

Assessing driver state is a substantial task, complicated by the various nuances and idiosyncrasies that characterize human behavior. Measurable components indicative of driver state may often reside in some high dimensional feature space [7]. Researchers have found that SVM have been particularly useful for binary classification problems. SVMs offer a robust and efficient classification approach for the problem of lane change detection because they map the driving data to high dimensional feature spaces where linear separating hyperplanes are sufficient to separate to two categories of data points. A correct choice of kernel and data representation can lead to good solutions. The problem of choosing the correct kernel is not trivial. For the purpose of this work, different types of kernels were employed to compare their performances at the classification task. However more sophisticated techniques now exist to allow systematic kernel estimation [18]. The benefits of SVMs over using HMM based methods are described later when the HMM framework is explained in detail.

2.3 Kernel Selection and Data Representations

A key issue in using the learning techniques is the choice of the kernel K in Eq (1). The kernel $K(x_i, x_j)$ defines a dot product between projections of the two inputs x_i and x_j in the feature space (the features been $\{\varphi_1(x), \varphi_2(x), \dots, \varphi_N(x)\}$ with N the dimensionality of the RKHS). Therefore the choice is closely related to the choice of the “effective” representation of the data, e.g. the image representation in a vision application. The problem of choosing the kernel for the SVM, and more generally the issue of finding appropriate data representations for learning, is an important one [16]. The theory does not provide a general method for finding “good” data representations, but suggests representations that lead to simple solutions. Although there is not a general solution to this problem, recent experimental and theoretical work provides insights for specific applications [16, 19, 20, 21]. However recent work from researchers [18] has shown that for a given data representation there is a systematic method for kernel estimation using semi-definite programming. Estimating the right kind of kernel remains an important segment of the future work with this kind of work. Once the right kind of kernel that best learns the available data is known the classification can be more accurate. At this point of time the available data was tested against different types of kernel functions to know the performance of each of them experimentally. Some of the kernel-types tested were linear, polynomial, exponential and gaussian. However, it was observed that all the kernels performed as good as or worse than linear kernel, as will be reported in the results. All the final results with SVM classification are therefore analyzed with linear kernel.

2.4 Training Support Vector Machines for LC/LK

The following few sections describe the issues with using support vectors machines for lane change detection and the solutions that were suggested.

2.4.1 Choosing a Window of Time

One issue with using SVM for lane change detection was that lane changes do not have fixed time length. Lane changes vary anywhere between 1 to 5 seconds. Direct temporal mapping between the data and SVM classification is not possible. Longer lane changes see a smooth transition in features values like steering angle, lane position, acceleration, etc. whereas shorter ones have a relatively abrupt transition.

Driving data is highly temporal in nature - that is the feature values change as a function of time. Moreover, one feature may be a function of one or many other features. The exact inter-dependency between features or within features themselves is not clear. In the domain of detecting driving maneuvers like lane changes, the change in the features with respect to time and their inter-dependency is more critical than the individual values of the features. For example, studies have shown that during a lane change drivers exhibit an expected sine-wave steering pattern except for a longer and flatter second peak as they straightened the vehicle [15]. The figure below shows such a pattern of steering against time.

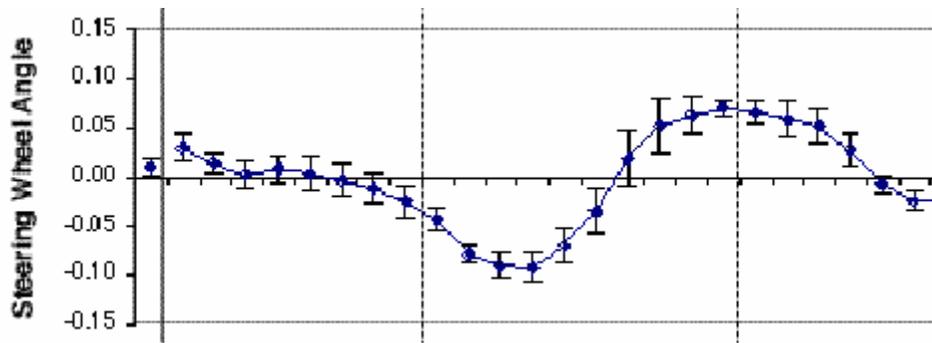


Figure 6: Steering Angle Displays a Sine-wave like Pattern (Source: Salvucci & Liu,'02)

Drivers first steer toward the destination and then back to center to level the vehicle on a steady path to the destination lane. Then, in a continual smooth movement, drivers steer in the other direction and back to center to straighten the vehicle in the destination lane.

Such patterns can only be observed (by humans) or learned (by machines) when a reasonable sized window of samples is observed. Thus for all practical purposes when training the SVM for a lane change or lane keeping an entire window is input instead of discrete samples.

To pick up training samples, the entire stream of training data is broken down into fixed size smaller windows. Each window corresponds to a training example. The windows can also overlap with each other. The size (time length) of the window that adequately captures the patterns within the features is a free parameter and therefore left to experimentation. Various window sizes were analyzed between 1 second to 5 seconds. The results with different window sizes are reported at the end of this chapter. A window size of 1.2 seconds (~12 samples) works well to train SVM.

Another key issue in modularizing the data stream into smaller windows is how to label each window. Using the definition of lane change (explained in section# 1.2.1), each sample on the training data can be labeled positive (LC) or negative (LK). However, a single training example consists of multiple samples, each sample with its own label.

The last sample in a window offers the latest information which is also used to label the entire window. Thus if the last sample in a window of size N samples is positive (LC), the entire window is marked positive and vice versa. But most importantly, we are most interested in the last sample as the most recent one. One must remember that to predict a classification label for any sample only the preceding samples can be used. Thus the last sample is the best indication of what the driver is currently trying to do.

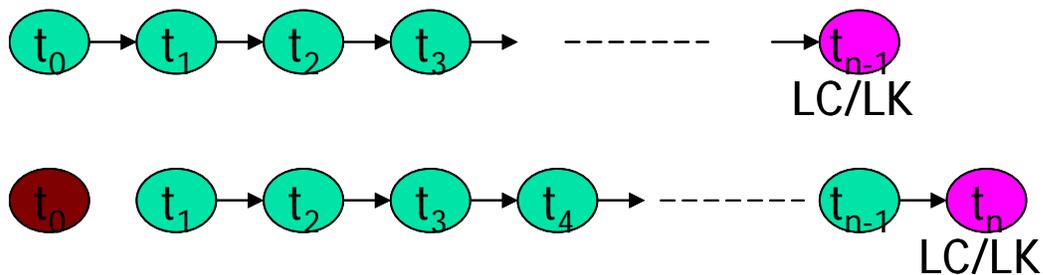


Figure 7: Moving Window of Constant Size

As shown in Figure 7 a single window of size N is defined by N samples at times $\{t_0, t_1, \dots, t_{N-1}\}$. The label of sample at t_{N-1} is used to classify the entire window. A moving window is used as shown in the figure. Whenever a new sample is obtained, it is added to the moving window and the last sample is dropped thus maintaining a constant size window.

2.4.2 Choosing a Data Representation

As argued earlier, the problem of data representation is an open one. However, good solutions with SVM depend significantly on the representation and kernel selection. A simple approach to data representation is to input the entire training window to the SVM with the actual values of the features [7].

In such an approach, a single window of size N samples is defined as

$$[\text{steer_angle}(t_0), \dots, \text{steer_angle}(t_{N-1}), \text{speed}(t_0), \dots, \text{speed}(t_{N-1}), \dots]$$

in general is equivalent to

$$[F^1(t_0), \dots, F^1(t_{N-1}), F^2(t_0), \dots, F^2(t_{N-1}), \dots, F^M(t_0), \dots, F^M(t_{N-1})]$$

where $F^x(t_i)$ represents the value of feature F^x at time t_i . Such a vector was used to train Relevance Vector Machines (RVM). RVM is a probabilistic sparse kernel model identical in functional form to the SVM [8]. Embedded in this formulation is the fact that temporal variations in maneuver execution are handled implicitly by RVMs. However, inherent functionalities of RVMs would fail to observe any dependencies or relationship between $\text{steer_angle}(t_0)$, $\text{steer_angle}(t_1)$, $\text{steer_angle}(t_2)$ and so on which could be critical. Also this formulation results in abnormally long sized input vector leading to additional computational complexity.

An alternative approach is suggested to explicitly include some form of dependency/relationship measure between feature values rather than the original values.

As argued previously it is the change pattern in the feature values which is more critical than the values themselves. Variance of a feature over all the samples in the block was used to replace the original values of that feature.

Variance of a feature is given by

$$Var[x] = \frac{1}{N} \sum_{i=1}^N (\mu_x - x_i)^2$$

where N is the window size (number of samples), μ_x is the mean of the feature x within the window and x_i is the feature value of the i^{th} sample. Thus variance effectively captures the change in the feature values which is very critical to learn specific patterns. Variance of features is particularly useful in reducing the side effects of any noisy data to a good extent. Another reason that encouraged the use of variance is the reduced feature set that was used for final training which is explained in the following section.

Figure 8 explains the two data representations that were experimented using variance. A single window of size N is shown on the left hand side of the two representations where each feature F^x has N values. In Data Representation I (non-overlapping), a single window is divided into two equal halves and the variance of features from each half is used. Thus for every N values of a feature we obtain two values of variances from the first and second half of the window. The rationale for splitting a single window in two halves was the need to capture multiple change patterns within a window. For example, features like lane position or steer angle might change multiple times within a window but a single variance across the window will reflect only the

overall change. This observation resulted in several modifications that helped capture changes in the feature values at multiple levels.

Data Representation II (overlapping) shown in the figure uses a similar structure with the difference that the two halves overlap with each other. A window of size N is divided into three equal parts say a, b, c . The first half will consist of the first two parts (a, b) and the second half will consist of last two parts (b, c) so that part b is common to both.

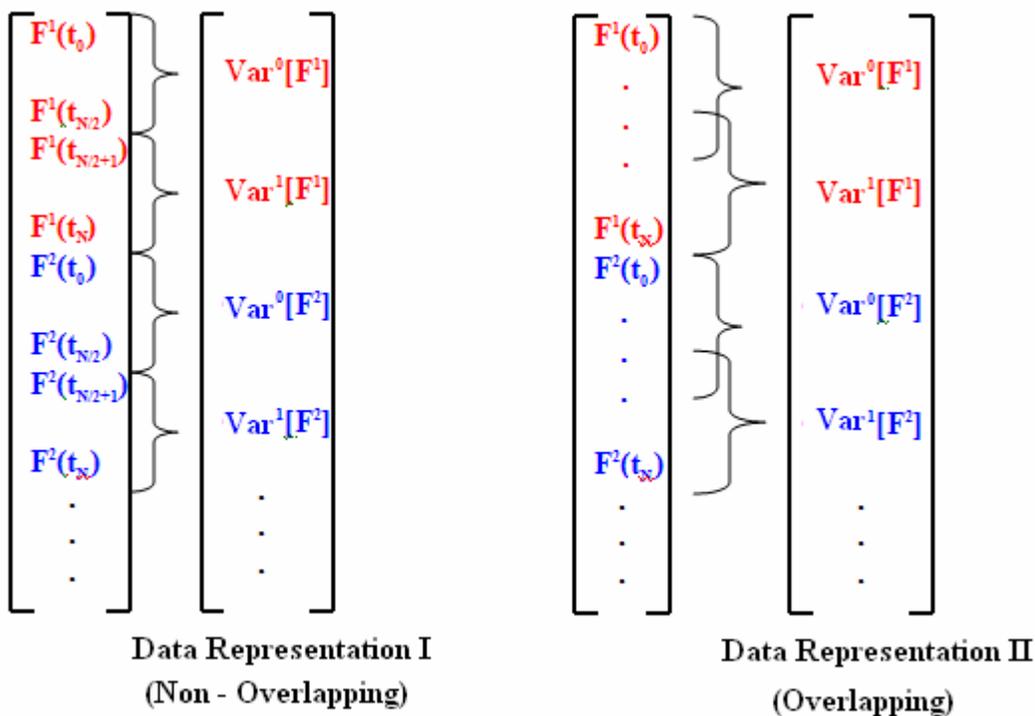


Figure 8: Data Representation

Overlapping structure was tested to account for the fact that the division of a lane change may not be equal and the changes may not happen always near the two ends of

the window. Experiments were performed with each representation and the results are listed at the end of this chapter.

2.4.3 Choosing a Feature-set

While training the SVMs it very important that only the most effective set of features is used. Features that display significant differences during a lane change against normal driving are the critical ones. Features that do not show enough predictability and vary randomly irrespective of a lane change should be avoided as they degrade the discrimination power of SVM.

With no prior preference to any feature, initial experiments included all features. Later, only selected combinations were employed to choose the minimal feature set that would produce the best classification. Various combinations of features were tested. However, only few selected combinations generated good results. Results with these selected combinations are presented later in this chapter. Best classification results were obtained with only four features with lane positions at different distances. Such an outcome was expected since lane position demonstrated the most consistent pattern among all the other features. One can argue that steering angle should also be a strong candidate. However, steering angle displays similar patterns both during lane change and while driving through a curved road which led to high number of false positives.

The current method of feature selection is based purely on experiments. However, as a future study use of more systematic stochastic techniques like t-tests, recursive feature elimination, and maximum likelihood test are planned.

2.4.4 Generating Continuous Recognition

An effective solution to simulate continuous recognition using discrete recognition is suggested. Classification label obtained for a single block ($t_0 - t_N$) would essentially mean that all the samples within that block can be classified under that label. More importantly it suggests the classification label of the last sample (t_N) in the block since all the previous samples ($t_0 - t_{N-1}$) serve as history data for it. To simulate a continuous recognition scheme we use a moving window of size N (block-size) by one sample at a time across the data to obtain classification labels for each sample. The concept of moving window is the same as one used while training. Consistency among classification scores is one important advantage of this scheme. That is if the previous and next few samples are classified positive the probability of the current sample to be classified negative is very low.

2.5 Results

While training the SVMs the most important factors that affected the results were (1) Data representation, (2) Window Size, and (3) Feature Set. Experiments were conducted with different configurations of each of these factors to obtain the best recognition system. Table 1 lists the different combinations of feature sets used. Table 2 and Table 3 show the recognition results using data representations I and II respectively

for different window sizes. The results denote the percentage of true positives detected in a ROC curve [28] at 1% and 5% false positive ratios. It is clear from the table that the recognition system is very robust and that almost all combinations produce good results through the two tables. The combinations that generated best results at 5% and 1% false positives are marked separately with red and green boxes respectively. Please note that although 5% false positives is our point of interest, it is interesting to note results at 1% also. Table 4 shows results with four different kernels using the best configuration obtained with. From the results it is evident that none of the kernels perform better than linear kernel at 5% false positives. Exponential kernel generates the best recognition at 1% false positives. For all final comparisons linear kernel is used since it generates the best result at 5% false positives. Further discussion of the results can be found in Chapter 4 where the three frameworks are compared together.

Table 1: SVM Feature Set

	Features
Set 1	Acceleration, Lane position 0, Lane position 30, Heading
Set 2	Acceleration, Lane position 0, Lane position 30, Heading, Lead Car distance
Set 3	Acceleration, Lane position 0, Lane Position 20, Lane position 30, Heading, Longitudinal acceleration, Lateral acceleration
Set 4	Acceleration, Lane position 0, Lane position 30, Heading, Steering Angle
Set 5	Lane position 0, Lane position 10, Lane position 20, Lane position 30

Table 2: Percentage Detected at 1 and 5 % False Positives (non-Overlapping)

Win Size	Set 1		Set 2		Set 3		Set 4		Set 5	
	1%FP	5%FP								
5 sec	50.17	83.53	50.28	90.03	50.29	91.16	50.12	89.96	54.87	91.13
4 sec	37.37	88.10	36.73	91.30	41.62	92.50	38.99	91.50	48.92	92.20
2 sec	50.13	89.30	50.33	93.00	53.30	97.70	44.71	94.00	55.22	97.40
1.5 sec	38.75	85.20	38.66	93.70	45.03	96.30	31.71	93.20	53.30	97.70
1.2 sec	59.46	96.83	59.51	96.00	58.90	96.00	58.42	96.05	67.48	96.70
0.8 sec	37.89	86.28	29.01	91.83	46.20	90.00	38.23	86.62	56.79	94.57

Table 3: Percentage Detected at 1 and 5 % False Positives (Overlapping)

Win Size	Set 1		Set 2		Set 3		Set 4		Set 5	
	1%FP	5%FP								
5 sec	47.85	87.1	47.95	86.93	48.89	89.00	49.23	88.07	49.58	87.77
4 sec	47.82	89.92	47.91	90.70	49.06	91.50	47.02	89.50	54.05	91.06
2 sec	56.26	96.20	54.60	96.16	58.12	97.90	49.20	95.79	61.20	97.32
1.5 sec	50.13	94.52	49.77	94.56	53.30	97.56	43.75	93.28	55.22	97.50
1.2 sec	39.25	93.79	38.77	93.70	45.03	95.00	31.71	93.20	53.07	97.94
0.8 sec	58.99	97.00	59.69	95.50	58.90	96.00	57.82	96.42	67.48	96.70

Table 4: Final Results with Different Kernels

	Linear		Polynomial		Exponential		Gaussian	
	@1%	@5%	@1%	@5%	@1%	@5%	@1%	@5%
1.2s, Set 5, Overlapping	53.07	97.94	58.12	97.94	69.40	97.17	51.08	97.40

2.6 Software Package for SVM

The WinSVM software available at www.kernel-machines.org was used for the purpose of my experiments with SVMs. It is windows version called the ‘SVM–light’ (<http://svmlight.joachims.org/>). The software offers an easy interface to use SVMs on a Windows platform. It also allows use of different kernel functions including a user defined kernel function. Further details on the software application package and the details on how to use it are described in detail in the Appendix.

3. Hidden Markov Model Recognition

Previous studies have shown that human behavior can be observed as a sequence of internal ‘mental’ states each with its own particular behavior and transition probabilities [22]. In case of driving, driver maneuvers like lane changing, turns, driving through a curved road etc can be seen as a sequence of internal mental states. For example a lane change can consists of following steps (1) a preparatory action centering the car in the current lane, (2) looking around to make sure the adjacent lane is clear, (3) steering to initiate the lane change, (4) the change itself, (5) steering to terminate the lane change, (6) a final re-centering of the car in the new lane. However, it is also observed that the internal states within the human mind are not directly observable and thus they must be determined by using some indirect estimation process. Hidden Markov model (HMM) is a superior method for recognizing temporal data patterns that can be expressed as stochastic transitions among finite discrete states.

This section reviews the second machine learning technique for detecting driver intentions. Hidden Markov Models (HMMs) have received burgeoning attention in the past few decades as a rich tool for modeling real-world signals. Its chief application is in machine recognition of speech.

3.1 Brief Introduction to HMM

Hidden Markov models represent an extension of simple, or observable, Markov models. A simple Markov model has three components: a set of states, an observation (i.e., physical event) associated with each state, and a set of transition probabilities between states. The model represents “fully observable” Markov processes in that there is a direct and obvious mapping between observations and states. HMMs extend Markov models by allowing each state to produce different observations according to probabilistic distributions. HMMs thus represent “hidden” Markov processes in that there may be numerous mappings between observations and states – that is, the state sequence is “hidden” from observation.

The basic three problems of the HMMs are as follows

1. Evaluation: Given a model and a sequence of observations, compute the probability that the observed sequence was produced by the model.
2. Decoding: Given the sequence of observations and model, determine the hidden sequence of states that is optimal in some meaningful sense.
3. Estimation: Given the sequence of observations and a model, adjust the model parameters so that the probability of current observation sequence given the model is the maximum.

Formal mathematical solution exists for each of these problems for HMMs, details of which can be obtained in [11]. Previous studies have found that driver behavior can be characterized as sequence of basic actions each associated with a particular state

of the driver-vehicle-environment [22]. Given sufficient training sequences, ‘estimation’ can be used to train model parameters adequately. Trained models can then use ‘decoding’ to determine the hidden sequence of states that corresponds to a particular driver behavior.

3.2 Motivation

There are two main motivations to use hidden markov models for the task at hand. First, HMM support recognition of temporal data patterns. This is particularly useful because humans perform different actions on a variable time-scale. Even within a lane change the internal states may vary in time. HMM provide an excellent framework for such temporal mappings. Second, human actions can be observed as some sequence of internal ‘mental’ states. However, this sequence is hidden. HMM can be trained to recognize this hidden sequence of states.

3.3 Developing HMM-based Recognition System

3.3.1 Choosing the HMM Structure

One of the issues in training HMM is selecting the right kind of HMM. Studies have shown that driving maneuvers like lane change often consists of a definite sequence of states. The time length for each internal state is unknown. This has encouraged the use of Left-right HMMs in previous studies [14, 22]. For consistency with previous work a similar structure was used in this work. Figure 9 below shows a left-right HMM. In such an HMM type, the model either can stay in a particular state or transition to the next state in the forward direction. The model can not skip states or transition to any state in the backward direction.

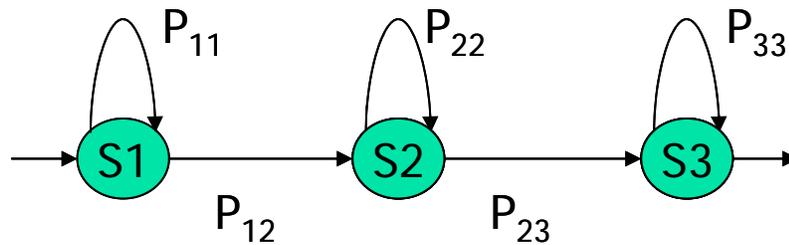


Figure 9: A Left-Right HMM

The data used for experimentation is the same that was used for training the support vector machines. The definition of lane change as explained under Section 1.2.1 was used for HMM system.

3.3.2 Analogy from Speech Recognition

The task of driving is seen analogous to speech. Different actions within the task of driving correspond to different words used in speech. The particular actions we are interested include (1) lane change and (2) lane keeping. In the domain of speech recognition words are considered as consisting of a sequence of sub-words or phonemes. For example the word ‘cream’ can be observed as a sequence of following phonemes,

‘cream’ \rightarrow k - r - iy - m

Lane change or lane keeping as already studied is a sequence of internal mental states. This analogy works well to match the temporal variations in the driving data and sequential nature of the internal states.

3.3.3 Training HMM for LC and LK

Following the analogy with speech recognition the HMM system was constructed to train three kinds of words. These are list in the table below.

Table 5: Word Dictionary and Structures

Words	Structure I (2 sub-words)	Structure II (3 sub-words)
1. Lane Change Left (LCL)	lcl1 – lcl2	lcl1 – lcl2 – lcl3
2. Lane Change Right(LCR)	lcr1 – lcr2	lcr1 – lcr2 – lcr3
3. Lane Keep (LK)	lk1 – lk2	lk1 – lk2 – lk3

To achieve recognition on a more continuous scale, each of the above words (LCL, LCR, and LK) were broken down into 2 or 3 smaller sub-words or phonemes. Each of these sub-words consists of 3 states each. Thus the recognition is performed not at a word-level but at a sub-word level. This also requires that we have a separate HMM for each of the sub-word instead of having a single HMM for one complete word. Figure 10 below shows the configuration corresponding to a lane change left (LCL) word with structure I. The structure is similar for other words. Note that the use of this structure is inspired by a similar sub-HMM structure used by Kuge et al. (2000). Experiments were performed with both structure styles and results are presented at the end of the chapter.

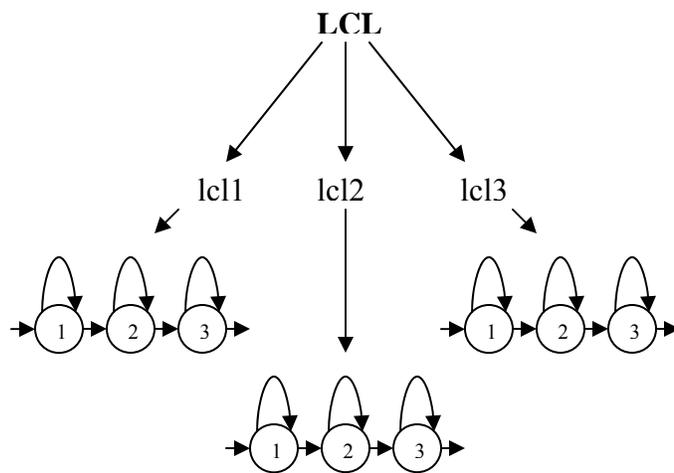


Figure 10: Configuration of 'LCL' Word (using Structure II)

To train HMM for a lane change (LCL/LCR) all the positive samples from the start of a lane to the end are input. Figure 11 below shows the window that constitutes a lane change training example. For training the SVMs a constant block size was used for all instances irrespective of their original sizes. However HMMs provide built-in support for temporal variations allowing variable block sizes.

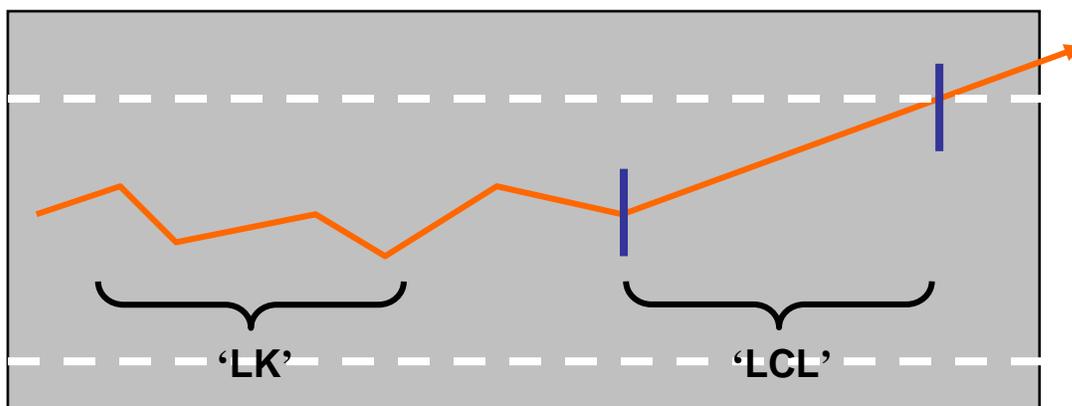


Figure 11: Training Windows for LC/LK

Some extremely rapid and/or degenerate instances of lane change resulted in very small size windows, smaller than the number of states in HMM. These can be referred to as abrupt lane changes. For such cases, the LC windows were padded with immediately preceding LK samples so that the total window size equals the number of states in the HMM. The block size should be at least as large as number of states because the HMMs are left-right HMMs with each state having transition to itself or the next state, thus at least one sample is required for each state. The LK training instances are not bounded by hard boundaries like LC instances and therefore are trained with constant block size. The window size for LK instances is a free parameter and experiments with different LK window sizes were performed to obtain the best value. Results with different LK window sizes are mentioned at the end of this section.

3.3.4 Choosing the Feature Set

One of the motivations to use probabilistic state machines like HMM was to capture temporally variant patterns in features like steering and acceleration. However it was possible that other features might also display patterns or dependency relation with other features. To identify the best set of features experiments were done with a set of most important features. Different combinations were experimented to reveal the most promising set of features. Results with different feature sets are listed in the concluding sections of this chapter. In future, systematic statistical tests need to be done for feature reduction.

3.4 Recognition

Once the sub-word HMMs are adequately trained using the training examples the recognition is a simple. We use a moving window of 7 seconds to generate what we call as ‘test sentence’ from our analogy to speech recognition. Figure 12 below shows the window that corresponds to a ‘sentence’. This is a large window of samples from the data stream. If for a given test sentence the last sample is a lane change sample the sentence is labeled as lane change. HMM uses decoding to determine the hidden sequence of states or words with the highest probability.

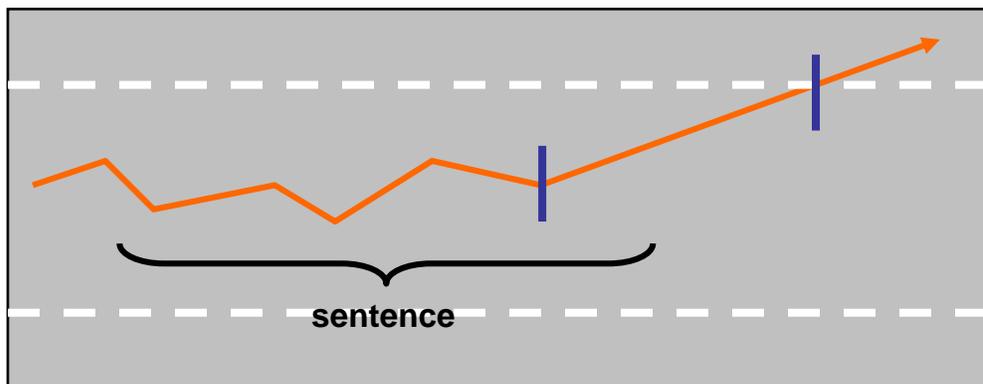


Figure 12: Input Test Sentence

The classification output for a sentence could be either a single word or a sequence multiple words. The last word in a sequence is considered as the classification output for the entire window.

For example, an output for a test sentence can be

$$\text{sentence} = \text{'LK'} + \text{'LCL'}$$

Such an output indicates that the model detected a sequence of two actions ‘lane keeping’ and ‘lane change left’ in that order. The last word in this sequence is important and signifies the overall classification label for the window. If the window was a lane change window and the last word was either ‘LCL’ or ‘LCR’, we conclude that the driver’s intention is inferred correctly.

3.5 Results

Training experiments were conducted with different sub-word structures, sizes for LK window and features. Table 7 and Table 8 show results with different combinations that were tested. Table 6 lists the different combinations of features used. Set 1 consisting of lane position values generated the best results for SVM. Set 2 consists of all lane positions and critical lateral features (heading, lateral acceleration, steering angle). Set 3 was configured by adding longitudinal features (speed, longitudinal acceleration) to Set 1. Set 4 finally had both lateral and longitudinal features along with lane positions. The configurations that generated the best results at 1% and 5% false positives are marked in green and red respectively. Note that Set 2, with a 2 second window worked best for 5% false positives while Set 4 with additional longitudinal features worked best at 1% false positive. Further discussion of the results is in the comparative study between different recognition frameworks.

Table 6: HMM Feature Set

	Features
Set 1	Lane position 0, Lane position 10, Lane position 20, Lane position 30
Set 2	Lane position 0, Lane position 10, Lane position 20, Lane position 30, Heading, Lateral Acceleration, Steering Angle
Set 3	Lateral Acceleration, Lane position 0, Lane position 10, Lane position 20, Lane position 30, Speed, Longitudinal Acceleration
Set 4	Lane position 0, Lane position 10, Lane position 20, Lane position 30, Heading, Lateral Acceleration, Steering Angle, Speed, Longitudinal Acceleration

Table 7: Percentage Detected at 1 and 5 % False Positives (HMM Structure I)

LK size	Set 1		Set 2		Set 3		Set 4	
	1%FP	5%FP	1%FP	5%FP	1%FP	5%FP	1%FP	5%FP
1.5 sec	40.52	67.84	35.17	75.10	41.86	66.25	37.26	73.52
2 sec	40.43	68.25	24.64	80.20	40.69	66.83	29.66	77.53
5 sec	41.85	66.92	41.35	75.02	43.44	66.00	43.80	73.77

Table 8: Percentage Detected at 1 and 5 % False Positives (HMM Structure II)

LK size	Set 1		Set 2		Set 3		Set 4	
	1%FP	5%FP	1%FP	5%FP	1%FP	5%FP	1%FP	5%FP
1.5 sec	47.02	63.58	43.19	72.43	47.20	62.32	47.54	70.93
2 sec	48.04	64.33	31.83	78.61	47.12	62.82	38.93	74.77
5 sec	49.02	63.83	50.31	69.51	47.95	61.24	51.96	68.84

3.6 HTK - Development tool

To develop this HMM-based recognition system, the HTK toolkit was used [12]. HTK provides an interface for speech recognition system and also facilitates USER/non-speech data files. Thus HTK could be used to develop HMM-based system for lane change detection. For details on the use of HTK please refer HTK manual available at [12] and Appendix.

4. Results and Comparative Study

This section compares the results of the two machine learning frameworks mentioned in this thesis with an existing one known as ‘Mind-Tracker’ [9, 10]. The mind-tracker was originally designed in a cognitive architecture called ACT-R to model and predict driver behaviors. Figure 13 below demonstrates the overall functioning of the Mind-Tracking system. The basic idea is to run many driver models in parallel each having its own intention about what to do next. The intention could be to change lane, take a turn, maintain in a lane etc. All these models are matched with the human model currently driving the car and the best matching model predicts driver behavior. For details on the Mind-tracking system readers are encouraged to refer to [9, 10].

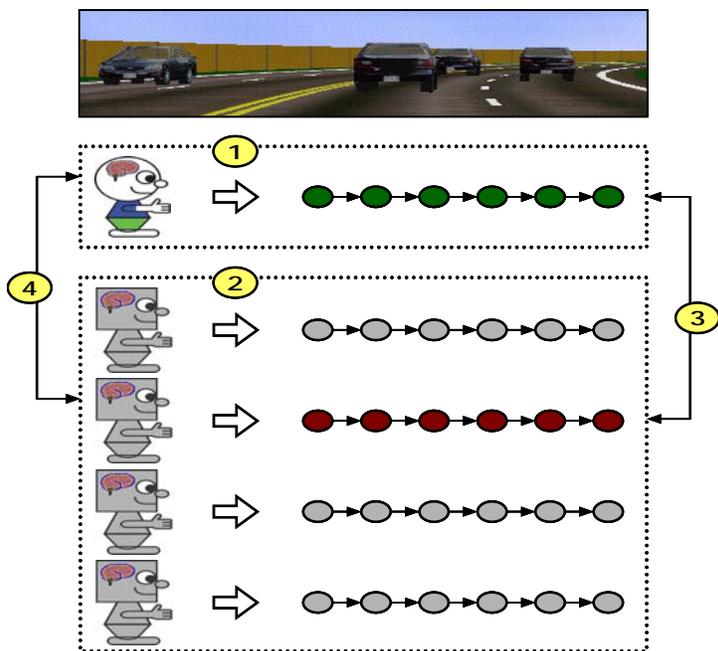


Figure 13: Mind-Tracker Overview

4.1 ROC curves

The mind tracking system uses the same data set as SVM and HMM in this thesis, which makes these comparisons possible. Figure 14 presents the ROC curves for a sample by sample recognition output from the three different frameworks. An ROC curve plots the true positives (Y-axis) against the false positives (X-axis). Perfect recognition would pull the curve through the point (0, 1).

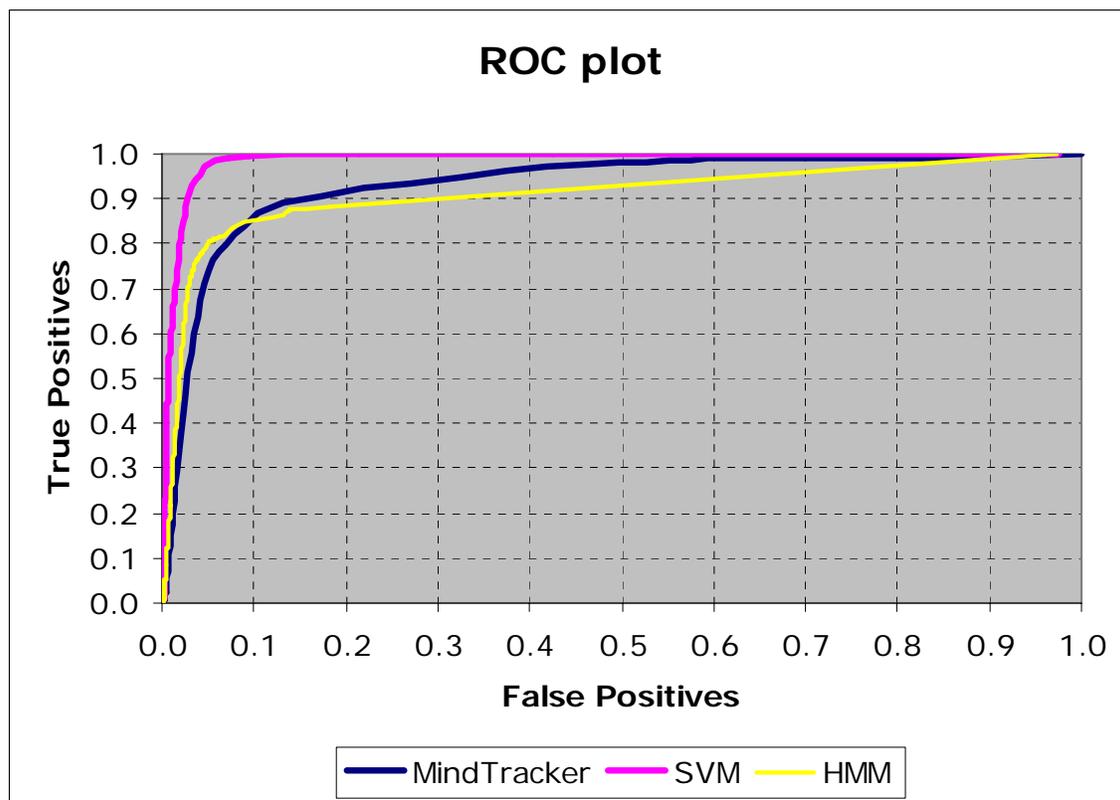


Figure 14: ROC Curves for the 3 Frameworks

The ROC curves clearly demonstrate the superiority of SVM recognition against the other two. At 5% false positives SVM generate a sample by sample accuracy of more

than 97% against 80% for the HMMs and 74% for the Mind tracker. This means that 97% of all the lane change samples were correctly detected by SVM while only 5% of the LK samples were detected as LC incorrectly. Experiments have shown that false alarm tolerance rate for human being is just about 5% and therefore the results are compared at that point.

4.2 Time Elapsed

Another way of looking at the recognition results is over time. Time is critical for these predictions and the sooner they predict lane changes the better is the system.

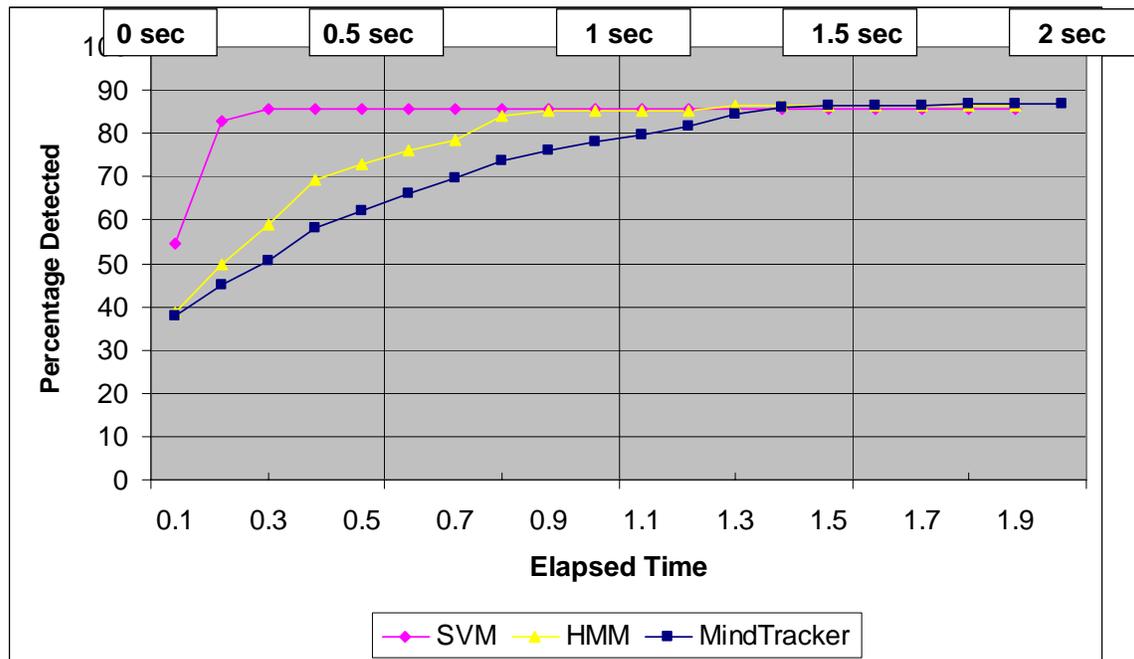


Figure 15: Time Elapsed Chart

Figure 15 show charts to demonstrate the time taken for each framework to detect a lane change. The time is calculated from the start of a lane change until the first sample in LC instance is detected positive Both HMMs and SVMs are consistent in their predictions which allow the assumption that once a sample is detected positive in an instance all the following samples are detected positive.

The above figure shows that SVM can correctly predict more than 85% of the lane change instances in less than 0.5 seconds (0.3 seconds) after the lane change starts. Both HMM and the mind tracking system detect about 72% and 62% respectively. Thus we conclude that SVM architecture is successful in inferring driver intentions very early in time. We note that at 1 sec both HMM and SVM produce 85% correctness against 78% percent for Mind-tracker.

4.3 Lateral Movement

A third way to analyze the results is to compute how much distance does the vehicle travel laterally before the lane change is detected. The lateral distance traveled by the car is calculated relative to the point where the lane change starts. Figure 16 shows comparative analysis for all three techniques.

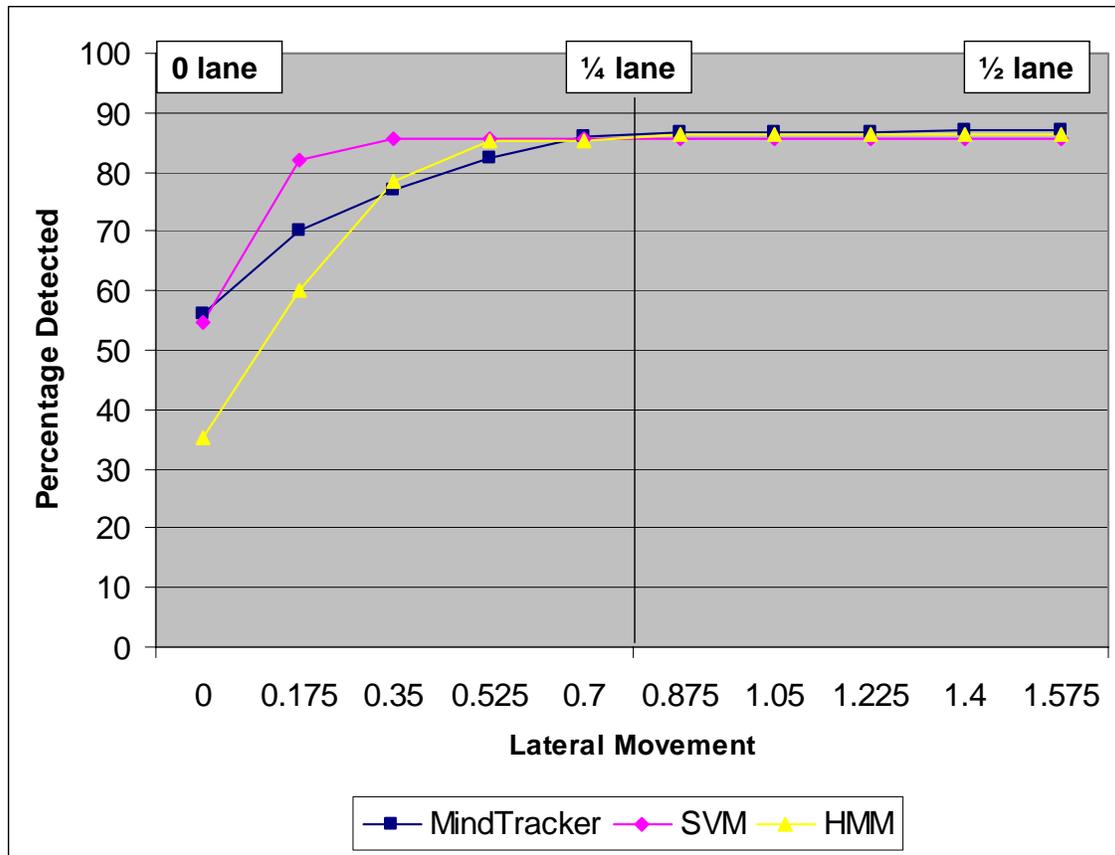


Figure 16: Lateral Movement Chart

SVM, HMM and Mind tracking system detects about 85% of the lane change instances before the car laterally moves 1/4 of the lane. In case of SVM we observe that it achieves about 86% accuracy even before the car moves 1/8 of the lane laterally. HMM prediction begins with a low accuracy initially but performs well at 1/4 of the lane and remains constant there after.

The following charts show comparative results from the three frameworks at essential points of comparison. On a sample by sample recognition scale the three frameworks are compared at 1% and 5% false positive rates. For the other two charts, the comparison is made for the percentage of instances correctly predicted at different time and lateral distances by each method. These results suggest that SVMs perform best as compared to the other two frameworks.

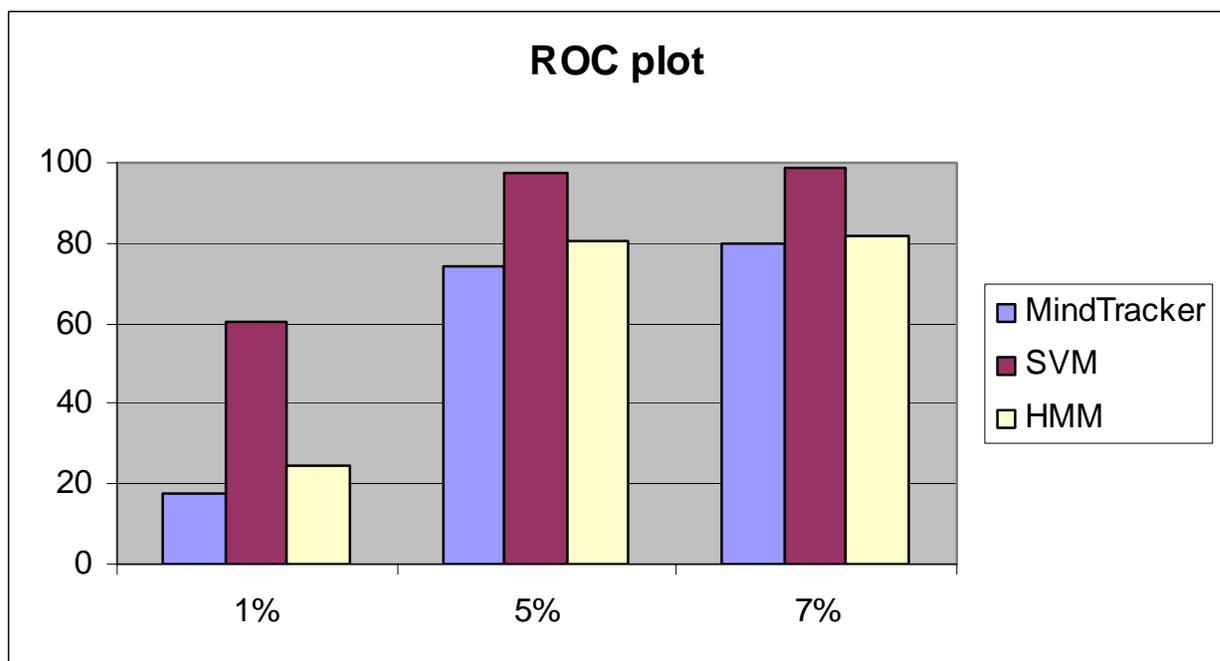


Figure 17: ROC Comparison Chart

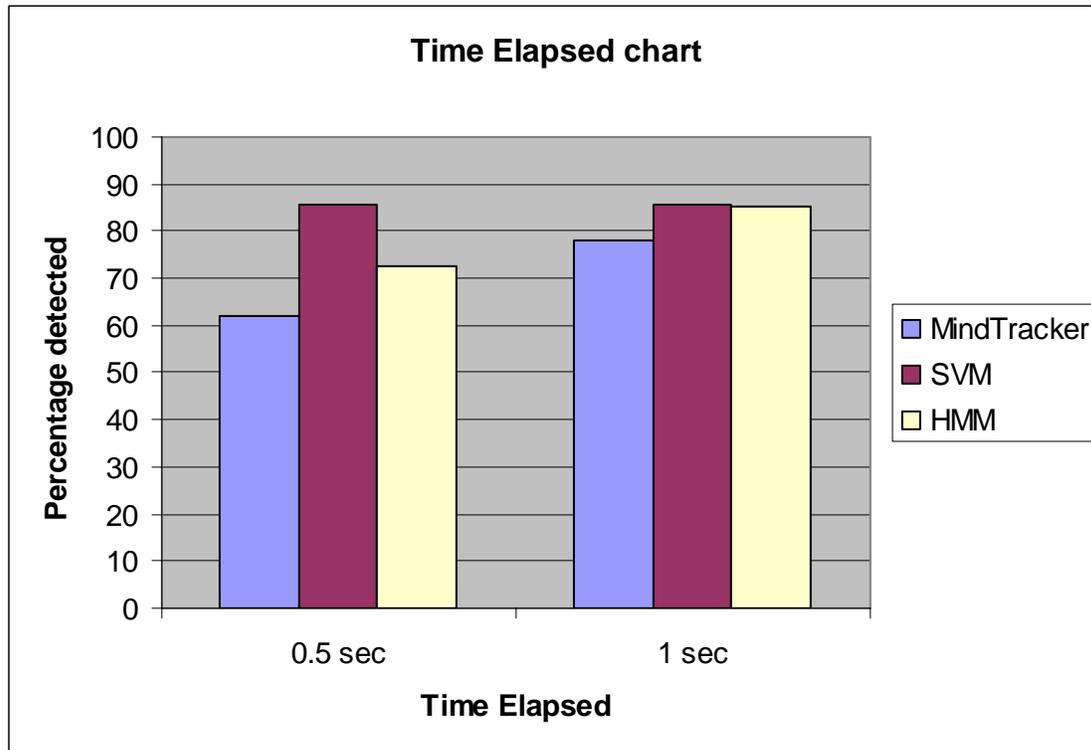


Figure 18: Time Elapsed Chart

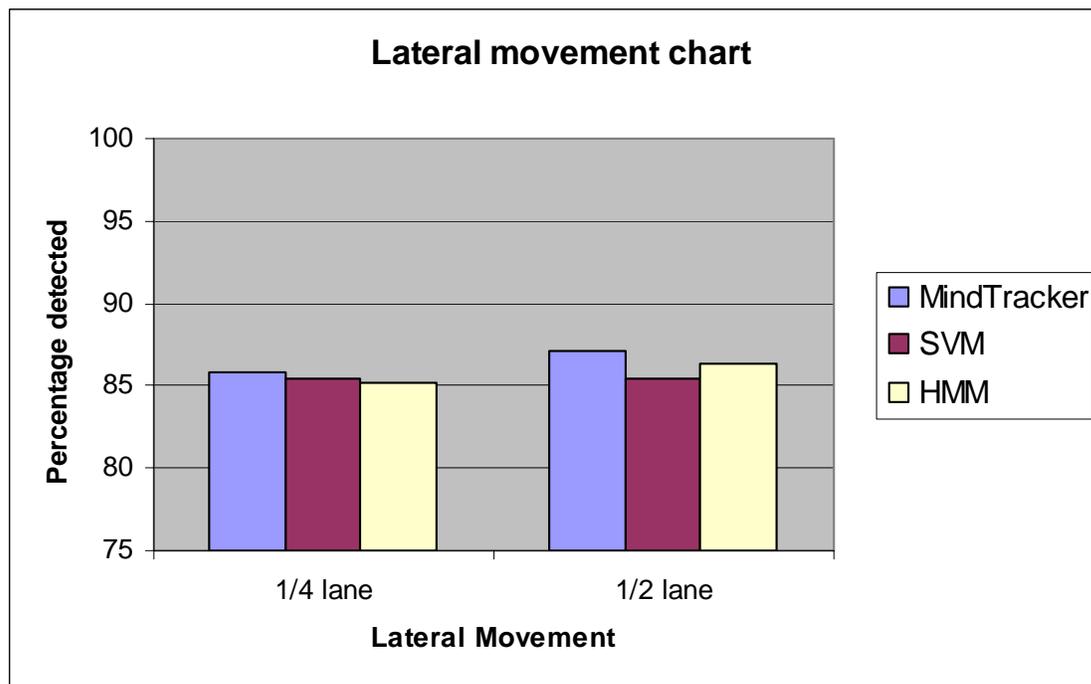


Figure 19: Lateral Movement Chart

5. Graph theory based Lane Change Detection

5.1 Introduction

The remaining section of the thesis presents a brief overview of the graph-based theoretic formulation of detecting lane change. This technique has only been partially explored and therefore has a special place at the end of the thesis.

Given the different features for a driving scenario like speed, acceleration, steering angle, etc. we study the correlation among these features during a lane change. A layered graph, with each layer having n nodes (n is the number of features) can be constructed. The edges in the graph go from one layer to the next and their weights represent the degree of correlation. Upon obtaining reference graphs for typical lane change instances we propose to match the input graphs with these reference graphs using efficient matching techniques.

The problem of lane change (LC) detection could prove vital for issues of driver security. Most algorithms used until now use the original form of data and try to study specific trends in the features over the interval of lane change instance. However none of the earlier algorithms used the fact that the dependency between two features could be significant. Two features could be correlated positively or negatively. Our approach utilizes the fact that some pairs of features display strong dependencies between each other during lane changes, a situation that does not occur during normal lane keeping (LK). More efficient and robust matching algorithms can take care of noise factor and time invariance of a lane change instance.

5.2 Overview of the Technique

5.2.1 From Continuous Stream of Data to Correlation Matrices

The input data is a continuous stream of values of the different features sampled at the rate of 10 samples per second. Depending on the maximum and minimum limits of these features we normalize each single value between [0,1]. The correlation coefficients between any pair of features is computed using these normalized values over a period of time. After studying the experimental data, it was observed that a time frame of 2 seconds would capture the dependency structure between the features most efficiently with minimal effect of noise in the data. Next significant change, if any, in the dependency structure was best observed at a shift of 1/2 second. Figure 20 shows the construction of first two correlation matrices from continuously varying samples of data. The correlation coefficient $CC[X, Y]$ between two features X and Y can be computed as follows

$$CC[X, Y] = \frac{\text{cov}[X, Y]}{(\sigma_X * \sigma_Y)}$$

where,

$\text{cov}[X, Y]$ = covariance between feature X and Y,

σ_X = standard deviation of feature X,

σ_Y = standard deviation of feature Y.

The values of $CC[X, Y]$ lie between [-1, 1]. A negative correlation coefficient represents that when one feature increases, the other decreases while a positive

correlation coefficient represents change in the same direction. Thus given values of these features over a given period of time we can compute the correlation matrix.

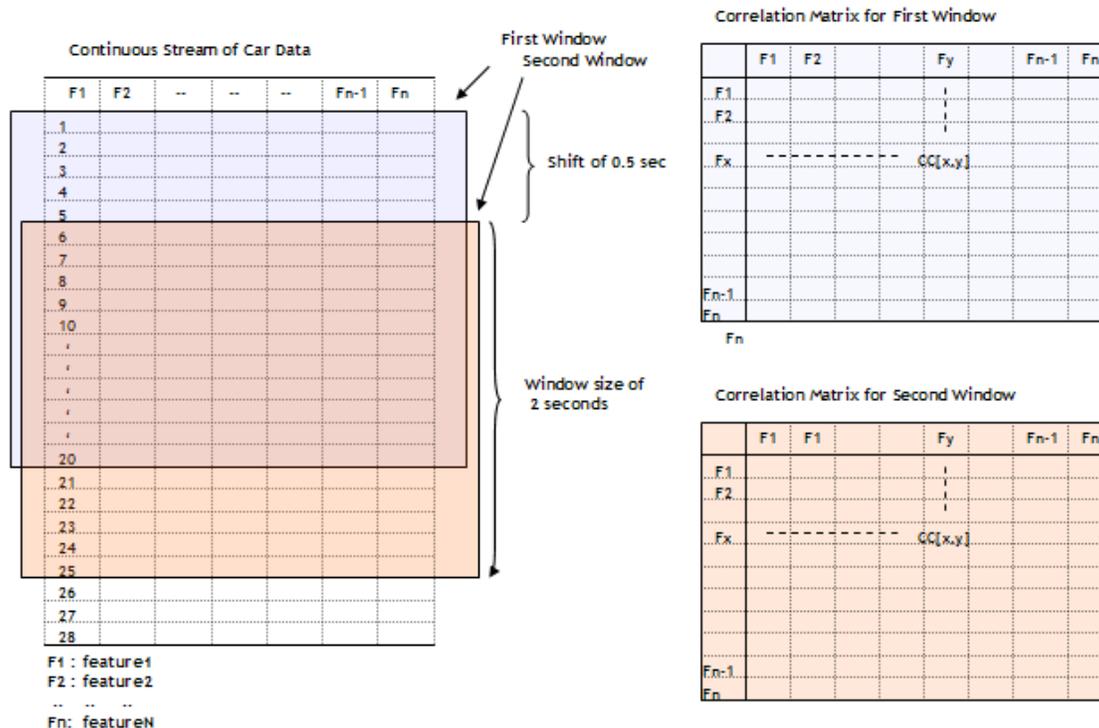


Figure 20: Continuous Stream to Matrices

5.2.2 From Correlation Matrices to Graphs

A single correlation matrix will represent one layer in the graph as shown in Figure 21. An edge from node X of one layer to node Y of the other will have a weight given by the $CC[X, Y]$. Next layer in the graph is built from the second correlation matrix obtained with a shift of 1/2 second from the previous window. Note that the correlation matrix is symmetric i.e. $CC[X, Y] = CC[Y, X]$. Therefore we consider only the upper triangle of the matrix resulting in only downward pointing edges in the graph.

Thus progressively we can compute layers of our graph by shifting window by 1/2 second and calculating the correlation matrices.

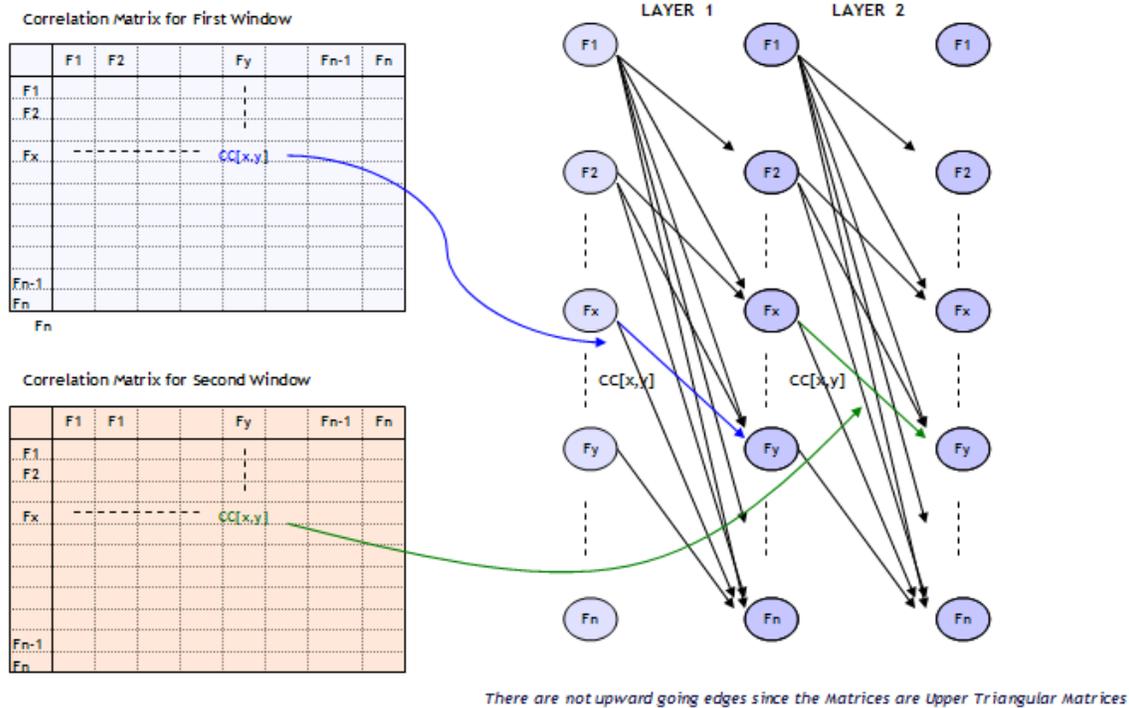


Figure 21: Matrix to Graphs

5.2.3 Construction of the Database

We need to construct reference graphs that would represent a typical lane change. We could classify all the instances of lane change into few patterns based on the direction of turn (say, left-to-right or vice versa), distance of lead car, acceleration etc. For example, one sample pattern- say pattern ρ - would be change of lane from left to right when the car/truck in the front is going slow and the right lane is faster. The driver tends to accelerate to catch up with the speed in the right lane, close in with the slow car/truck in the front and then drift towards the other lane. Another example would be the case

when both lanes are equally fast but driver needs to change lane to take either a turn/exit/merge. Each such pattern would have a reference graph in our database.

A lane change would last for about 4 to 5 seconds maximum from the time when the driver starts his final drift towards the other lane until he crosses the lane. To construct a reference graph for a particular pattern of lane change last 4.5 seconds of data before a lane change was observed. 4.5 seconds of data with a window size of 2 seconds per layer and shift of 1/2 second for the next layer would give us 7 layers in the reference graph. Figure 22 shows the construction of reference graph of a pattern ρ from all the instances of that pattern found in the experimental data. To calculate an average correlation matrix for a layer we list together corresponding windows from all the instances of pattern ρ to give a single window. The correlation matrix derived from this window would represent a corresponding layer in the reference graph. Thus, this layer so obtained is representative of all the instances of that pattern of lane change. The dependencies so obtained between features would be typical of pattern ρ . Figure 22 shows construction of first two layers of the graph.

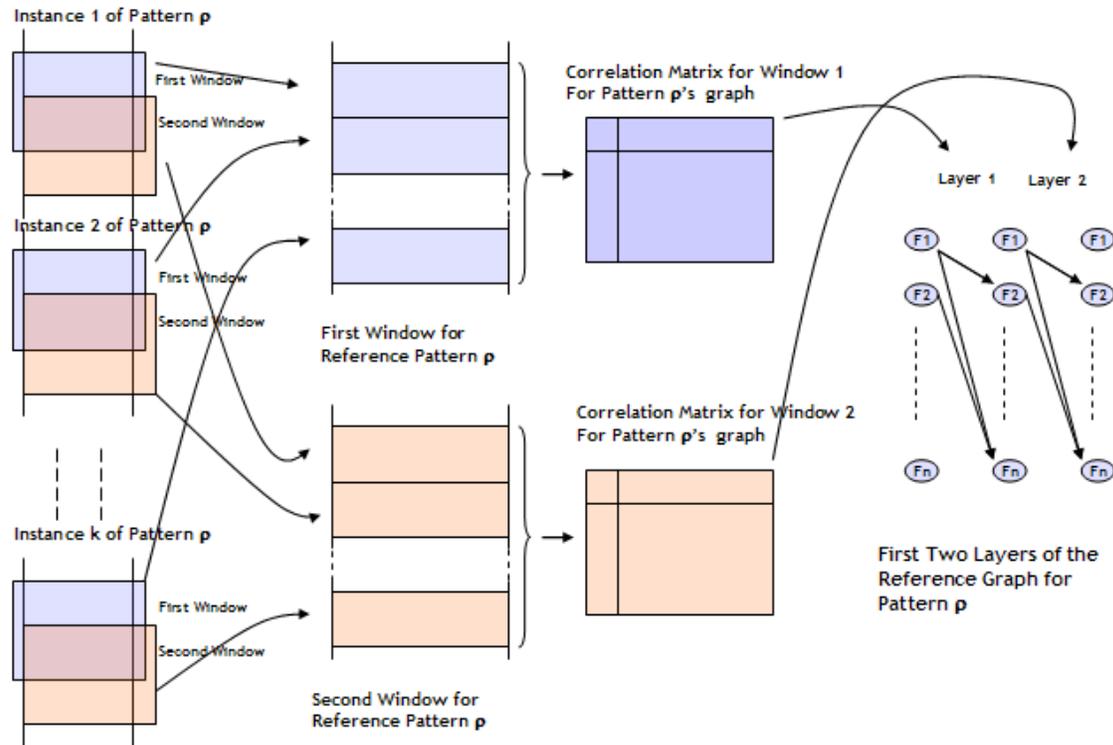


Figure 22: Construction of Database

5.2.4 Simple Graph-Matching Algorithm

Figure 23 shows the general lane change detection strategy. Input graph is matched against pre-stored database of graphs using matching and data mining techniques to differentiate between LK and LC.

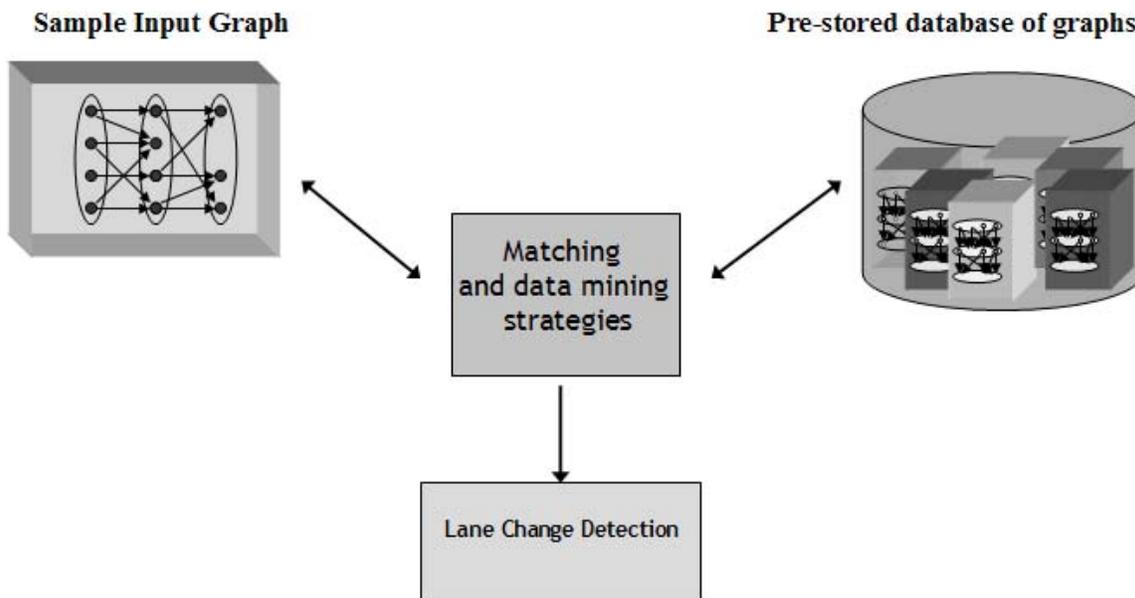


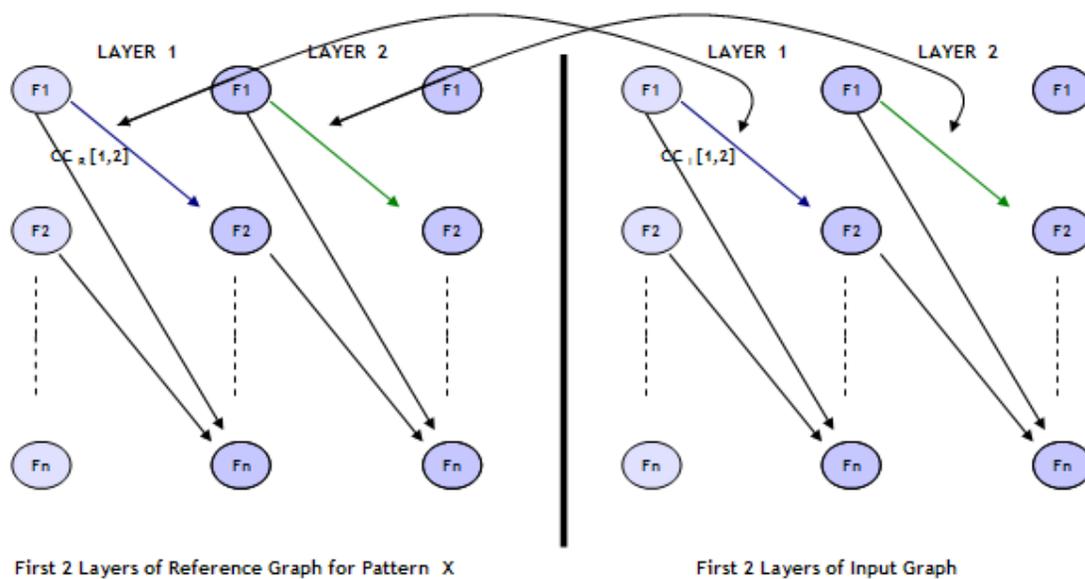
Figure 23: Matching and Data Mining Strategies

A simple matching algorithm between two graphs maintains all the edges between two layers irrespective of their weights and performs a one-to-one match between edges. A total of n features would give us $n*(n-1)/2$ edges going from one layer to the other. In a complete reference graph (all 7 layers) with no edges removed let m be the total number of edges. We compare weights between corresponding edges. Figure 24 shows the simple matching algorithm. Let $CC_I[X, Y]$ be the weight of edge $[X, Y]$ in input graph and let $CC_R[X, Y]$ be the weight of edge $[X, Y]$ in reference graph. We compute the degree of matching between these two edges as follows

Degree of matching = 0 if $CC_I[X, Y]*CC_R[X, Y] < 0$

= 1 Otherwise

Upon adding the degree of matching for each edge in a graph we get the degree of matching between two graphs by summing up for all the edges in the graph. This will be some value less than m . More than 70 percent matching i.e. if degree of matching between graphs $\geq 0.7 * m$ we would typically predict a possible lane change ahead in the input data.



$$\text{Degree Of Matching of Edge}[1,2] \text{ in Layer 1} = 0 \quad \text{if } CC_r[1,2] * CC_i[1,2] < 0$$

$$= 1 - \text{abs}(CC_r[1,2] - CC_i[1,2]) \quad \text{otherwise}$$

****Degree of Matching for the Input Graph is the sum of Degree of Matching of all the edges in the Graph**

Figure 24: Degree of Matching

5.3 Results

The input car data were provided by Nissan Motor Co., Ltd. The different features that were used for analysis were accelerator pedal pressure, acceleration, lead car distance, steering angle, steering torque, yaw-rate, car's longitudinal acceleration, heading, lateral position at 0 distance, lateral position at 30m, lateral acceleration, and steering angle velocity.

Our database consists only 4 reference graphs of very frequently observed patterns. Every input graph would be compared with each of these 4 reference graphs and a degree of matching was obtained against each reference graph individually. A typical input file would be a continuous stream of data consisting of instances of both LC and LK for a car over a sufficiently long period of time. The matching program would generate a 7-layered input graph for every interval of 4.5 seconds. The next input graph from the file would be obtained by a shift of 1/2 second interval. Essentially 4 seconds of data from the previous graph is reused. Thus a continuous stream of input graphs is fed into the matching program and input graphs with degree of matching higher than certain cutoff were screened and noted down.

Table 9 shows the percentage of success/failure in predicting lane changes with different cutoffs. More and more experiments on the input data indicated that the reference graphs constructed were more representative of LC patterns in general than any pattern in particular.

Table 9: Results with Graph Technique

CUTOFF %	LC instances correctly detected (%)	LC patterns correctly detected (%)	LC instances falsely detected (%)
65	75	20	11
70	60	12	1.2
75	22	6	0.001

At this point, we cannot compare the graph-based algorithm to the other three frameworks. This is mainly because the algorithm does not produce continuous recognition. Also the technique has only been partially explored and uses an oversimplified graph-matching algorithm. A continuous output can be produced by a sliding the window by one sample at a time. However, robust techniques such as those mentioned in the future work are required to build a promising recognition system.

6. General Discussion

As observed from the comparative analysis of the three different frameworks, it is evident that both SVM-based and HMM-based learning methods are successful in predicting driver actions. Both these techniques produce better results than the existing Mind tracking system in most aspects. The two frameworks have capabilities to produce continuous recognition on a sample by sample basis. Also as far as recognition is concerned, it occurs in real time in both these frameworks. The HMM-based technique does better in terms of early prediction when compared to the mind-tracking system. However the mind-tracking system does better when compared over lateral movement.

The SVM framework proved very effective because of high sample by sample prediction rate which is highly desired in a real time scenario. Results show a significant 23% better prediction accuracy compared to the mind tracking system. SVM have also demonstrated excellent accuracy in terms of early prediction with respect to both time and distance. More than 85% of the lane change intentions were detected in less than 0.3 seconds after the driver initiates his action. Thus it can be concluded that the SVM based framework not only offers accurate recognition capabilities for driver's intentions but also assures highly reliable prediction rate. Such a framework can improve the performance of a driver support system due to its early prediction.

Although the SVM recognition system has demonstrated good capabilities for early recognition the technique can be improved for robustness. Future work on the framework aims at stochastic techniques for feature selection such as t-test, maximum likelihood test or feature elimination method [21]. Such methods will offer a better

understanding to select the right feature set. Selecting the optimal feature set will enhance performance of both HMM and SVM recognition. In addition, systematic kernel estimation techniques need to be employed for to explore the best kernel space. Recent work on kernel estimation [18] suggests the use of semi-definite programming to estimate the right kernel. Complex binary search techniques can be applied to find the optimal variance pattern within a temporal window of lane change. This will allow the system to capture multiple patterns within the window more effectively.

The graph matching algorithm explained is oversimplified and is vulnerable to erroneous matching. Not every feature is dependent on every other to the same extent. It might be the case that two features are completely independent resulting in no edge between them while some features may have strong dependency between them. Weights of edges between two independent nodes (features) might match coincidentally adding to the degree of matching. Some simple modifications that would promise better results are as follows

- Deleting edges with weights close to zero.
- Classifying LC patterns more precisely and augmenting reference graphs in the database.
- Augmenting the database with reference graphs for LK instances and eliminate input graphs similar to LK patterns.
- Applying scaling factor to degree of matching between edges. Some edge weights in the reference graph display high contrasts between LC and LK instances. We might multiply degree of matching with a suitable scale to represent the contrast.

We intend to use more robust algorithms that might use the topology of the graph to find a match. Edge-wise matching for large database could be time consuming. Indexing on the graphs using different signatures like eigen-values of the incidence matrix or carter pillar-dimension could prove really significant. Robust graph matching algorithms like bipartite graph matching and edit-distance based matching could also be applied.

SVM and HMM have been independently successful in detecting driver intentions however a combination of the two frameworks may have a better answer. HMM is good at mapping temporal variations and SVM offer a powerful sample by sample prediction accuracy. A combined recognition system that can inherit the merits of these two frameworks may prove useful.

A fuller range of driver behaviors like turns, stops, parking etc. can be tested on the same frameworks to observe the recognition capabilities. The frameworks have proved very effective for the lane change intentions and may have a solution of other intentions too.

Driver safety is a critical issue and for the same reason inferring driver intentions is proving to be critically important. Machine learning techniques like those presented in this thesis have shown remarkable potential for very early and reliable prediction of driver intentions. Application of such techniques in real world automobiles will definitely take the issue of driver security to the next level of assurance.

List of References

- [1] Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine Learning*, 20:273-295.
- [2] Schölkopf, B., Burges, C., and Vapnik, V. (1995). Extracting support data for a task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery and Data Mining*. AAAI press, Menlo Park, CA.
- [3] Blanz, V., Schölkopf, B., Bühlhoff, H., Burges, C., Vapnik, V., and Vetter, T. (1996). Comparison of view-based object recognition algorithms using realistic 3d models. In c. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks – ICANN'96*, pages 251 – 256, Berlin. Springer Lecture Notes in Computer Science, Vol. 1112.
- [4] Schmidt, M. (1996). Identifying speaker with support vector networks. In *Interface '96 Proceedings*, Sydney.
- [5] Osuna, E., Freund R., and Girosi, F. (1997). An improved training algorithm for support vector machines. In *Proceedings of the 1997 IEEE Workshop on Neural Networks for Signal Processing*, Eds. J. Principe, L. Giles, N. Morgan, E. Wilson, pages 276 – 285, Amelia Island, FL.
- [6] Joachims, T. Text categorization with support vector machines. Technical report, LS VIII Number 23, University of Dortmund, 1997.
- [7] Wipf, D. & Rao, B. (2003). Driver Intent Inference Annual Report, University of California, San Diego.
- [8] Tipping, M. E. (2001). Sparse Bayesian Learning and the Relevance Vector Machine, *Journal of Machine Learning Research* vol. 1, pp. 211-244.
- [9] Salvucci, D. D. (2004). Inferring driver intent: A case study in lane-change detection. *In Proceedings of the Human Factors Ergonomics Society 48th Annual Meeting*.
- [10] Salvucci, D. D., & Siedlecki, T. (2003). Toward a unified framework for tracking cognitive processes. *Proceedings of the 25th Annual Conference of the Cognitive Science Society*. Mahwah, NJ: Lawrence Erlbaum Associates.
- [11] Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, Vol. 77, NO. 2.
- [12] HTK Speech Recognition Toolkit, <http://htk.eng.cam.ac.uk>
- [13] Oliver, N. & Pentland, A. P. (2000). Graphical Models for Driver Behavior Recognition in SmartCar. *Proceedings of the IEEE Intelligent Vehicles Symposium 2000*.

- [14] Kuge, N., Yamamura, T. and Shimoyama, O. (2000). A Driver Behavior Recognition Method Based on a Driver Model Framework. *Intelligent Vehicle Systems* (SP-1538).
- [15] Salvucci, D. D., & Liu, A. (2002). The time course of a lane change: Driver control and eye-movement behavior. *Transportation Research Part F*, 5, 123-132.
- [16] Evgeniou, T., Pontil, M., & Poggio, T. (2000). Statistical Learning Theory: A Primer. *International Journal of Computer Vision* 38(1), 9-13.
- [17] Vapnik, V. (2000). *The Nature of Statistical Learning Theory*. 2nd Ed., Springer, New York.
- [18] Lanckriet, G. R. G., Cristianini, N., Barlett, P., Ghaoui, L. E., & Jordan, M. I. (2004). Learning the Kernel Matrix with Semidefinite Programming. *Journal of Machine Learning Research* 5, pp. 27-72.
- [19] Jaakkola, T. & Haussler, D. (1998). Probabilistic Kernel Regression models. In *Proceedings of Neural Information Processing Conference*.
- [20] Mohan, A. (1999). Robust Object Detection in Images by Components. Master's Thesis, Massachusetts Institute of Technology.
- [21] Vapnik, V.N. (1998). *Statistical Learning Theory*. Wiley: New York.
- [22] Pentland, A. and Liu, A. (1999). Modeling and prediction of human behavior. *Neural Computation* 11, 229-242.
- [23] Federal Highway Administration (1998). Our nation's highways: Selected facts and figures (Tech. Rep. No. FHWA-PL-00-014). Washington, DC: U. S. Department of Transportation.
- [24] Ahmed, K. I., Ben-Akiva, M. E., Koutsopoulos, H. N., & Mishalani, R. G. (1996). Models of freeway lane changing and gap acceptance behavior. In J.-B. Lesort (Ed.), *Transportation and Traffic Theory*. New York: Elsevier Science Publishing.
- [25] Gipps, P. G. (1986). A model for the structure of lane-changing decisions. *Transportation Research – Part B*, 5, 403-414.
- [26] Finnegan, P. and Green, P. (1990). The time to change lanes: A literature review (Tech. Rep. No. UMTRI-90-34). Ann Arbor, MI: The University of Michigan Transportation Research Institute.
- [27] Talmadge, S., Chu, R. Y., & Riney, R. S. (2000). Description and preliminary data from TRW's lane change collision avoidance testbed. In *Proceedings of the Intelligent Transportation Society of America's Tenth Annual Meeting and Exposition*.

[28] Fawcett, T. (2003). ROC graphs: Notes and Practical Considerations for Data Mining Researchers. HP Laboratories, Palo Alto.

APPENDIX A

Table 10: Complete Feature Set

Feature	Description
Brake Pressure	Brake Pressure
BrakeSW	Brake SW
Curvature0	Road Curvature (radians)
Distance	Lead Car Distance (meters)
Heading0	Heading
laneDetectFail	Lane Detection Failure
LatG	Lateral Acceleration
Lane Position 0	Lane Position at 0 meter
Lane Position 10	Lane Position at 10 meters ahead
Lane Position 20	Lane Position at 20 meters ahead
Lane Position 30	Lane Position at 30 meters ahead
lcFlag	Lane Change flag
lcLeftRight	Lane Change Signal – Left or Right
lcPattern	Lane Change Pattern
lonAccel	Longitudinal Acceleration
Speed	Speed
Steering Angle	Steering Angle
Steer Rate	Steering Rate
Steer Torque	Steering Torque
TurnSigLeft	Turn Signal Left
TurnSigRight	Turn Signal Right
Vr	Steering Angle Velocity
Yaw rate	Yaw Rate

Please note that *acceleration* mentioned in the thesis can be easily derived from *speed* feature. The simple definition used here is as follows,

$$acceleration(t_1) = speed(t_1) - speed(t_0)$$

1. Using WinSVM

The WinSVM software available at www.kernel-machines.org was used for the purpose of my experiments with SVMs. The software offers an easy interface to use SVMs on a Windows platform.

WinSVM offers easy interface to both train SVMs and test them for binary classification schemes. Also it offers easy interface to change between different kernels like linear, polynomial etc. Each training sample consists of a vector of features. A training/testing file given as input to the WinSVM software has the following grammar and includes one sample per line of the file.

<class> .=. +1 | -1 | 0

<feature> .=. integer

<value> .=. real

<line> .=. <class> <feature>:<value> <feature>:<value> ... <feature>:<value>

The class label and each of the feature/value pairs are separated by a space character. Feature/value pairs MUST be ordered by increasing feature number. Features with value zero can be skipped. The +1 as class label marks a positive example, -1 a negative example respectively. A class label of 0 indicates that this example should be classified using transduction. The predictions for the examples classified by transduction are written to the file specified through the -l option. The order of the predictions is the same as in the training data.

In my experiments I use class labels with values +1 and -1, where +1 denotes a lane change (LC) sample and -1 denotes a lane keeping (LK) sample. Upon successfully training the SVMs, WinSVM gives out a model file (*.mdl) that stores in the values of the support vectors and other metadata like the number of training examples (both positive and negative), type of kernel used , kernel parameters if any.

This model file is used for recognition when test files are given as input. The recognition results are output in a results file (*.rsl). The results file contains the recognition output of one test sample per line. Each line consists of the distance of that test point from the hyperplane and the sign indicates whether that sample was classified as positive or negative. A negative sign indicates that it was classified as a LK sample and vice versa for positive sign. Thus higher the absolute value of the distance indicates higher confidence to predict class labels.

2. HTK Development tool

To develop the HMM-based recognition system, the Entropic's HTK toolkit was used. HTK provides an easy interface for speech recognition system but at the same time facilitates USER/non-speech data files. Thus HTK could be used to develop HMM-based system for lane change detection.

The current work uses *HERest* and *HVite* tools from the HTK tool kit. For details on the operation of these tools users are requested to refer to the HTK manual available on the official HTK website [12].

Steps in Training and Recognition: (may not include in actual thesis)

1. Creating raw HMM definition files that defines the number of the states in the HMM, size of the data stream per state, type of HMM (left-right etc)
2. Processing raw data files to select the LC and LK training instances into separate files (LK*.out & LC*.out)
3. Creating label/transition files (LK*.lab & LC*.lab) for each of the training instances.
4. Converting the text files created in step 2, 3 into binary format using MATLAB code. HTK recognizes only binary format for input training files.
5. Training HMM parameters within the definition files using all the training examples to estimate mean, variances and transition probabilities [*HERest*].
6. Creating test files using steps 2, 3 & 4 into binary format required by HTK.
7. Decoding test samples using the HMMs [*HVite*]

8. Analyzing the results by calculating the LC score from the log probabilities and plotting ROC curve.

LC Score is calculated as follows:

$$\text{LC Score} = \frac{\log P(\text{LC}) + \log P(\text{LK})}{\log P(\text{LC})}$$

