Learning to Predict Driver Route and Destination Intent

Reid Simmons, Brett Browning, Yilu Zhang & Varsha Sadekar

***Abstract*— For many people, driving is a routine activity where people drive to the same destinations using the same routes on a regular basis. Many drivers, for example, will drive to and from work along a small set of routes, at about the same time every day of the working week. Similarly, although a person may shop on different days or at different times, they will often visit the same grocery store(s). In this paper, we present a novel approach to predicting driver intent that exploits the predictable nature of everyday driving. Our approach predicts a driver's intended route and destination through the use of a probabilistic model learned from observation of their driving habits. We show that by using a low-cost GPS sensor and a map database, it is possible to build a Hidden Markov Model (HMM) of the routes and destinations used by the driver. Furthermore, we show that this model can be used to make accurate predictions of the driver's destination and route through on-line observation of their GPS position during the trip. We present a thorough evaluation of our approach using a corpus of almost a month of real, everyday driving. Our results demonstrate the effectiveness of the approach, achieving approximately 98% accuracy in most cases. Such high performance suggests that the method can be harnessed for improved safety monitoring, route planning taking into account traffic density, and better trip duration prediction.**

I. INTRODUCTION

Much of our driving is routine in that we tend to go to the same destinations over and over, following the same routes at the same time of day, or day of week. Even when better routes exist, such as shorter, or faster routes given the current traffic conditions, we tend to stick with routes that we have used in the past. It is this observation that motivates the ideas presented in this paper.

The problem we focus on in this paper is to develop a system that collects data on the habits of individual drivers by observing what destinations they drive to and what routes they take to get there. Using that data, the system should be able to predict the driver's intended route and destination based on what has been observed of the driver's route taken, so far.

There are several possible uses for such types of predictions. In terms of navigation systems, the predictions could be used to provide better route guidance without requiring input from the driver and instead relying on the prediction to infer the driver's intent. Smarter route guidance could be provided through the integration of real-time traffic estimates. Alternatively, the vehicle could estimate the time of arrival at the predicted destination and alert the driver if she is going to be late (by accessing calendar information stored on the drivers PDA). Another application area is safety warning, where knowledge of the driver's intended short-term route can be used to reduce false alarm rates.

In this paper, we present a Hidden Markov Model (HMM) based approach to providing real-time predictions on driver destination and route. Our approach is based on building the probabilistic model through observation of the driver's habits via a map database and a low-cost GPS sensor. Our results demonstrate that this approach can achieve significant levels of accuracy on real, everyday driving data.

The paper is structured as follows. In the ensuing section we describe the basis of our approach and provide the mathematical machinery required to understand the HMM approach. We then describe how we build a HMM for learning and predicting driver routes and destinations. Based on this model, we present the experimental results using a corpus of real driving data. We then conclude the paper.

II. THE APPROACH

The fundamental assumption in this project is that driving is largely routine, and that past performance can be used to predict what the driver will do in the future. We further assume that a route map is available as is a sensor, such as GPS, that can tell us what segment of the map the vehicle is on, what speed the vehicle is traveling, and the time at which the sensor reading is taken. Additional information (such as turn signal state, which lane the vehicle is in) may be useful, but has not been explored in this project, to date.

While one would like to make perfect predictions, this is not possible due to the nature of driving. Even if a driver has very set routines, it is still possible for the driver to deviate from them once in a while. For instance, if a driver always goes to work at 9AM on weekdays, using the same route every time, once in a while she may go somewhere else at that time, such as a doctors appointment. Alternatively, a driver may have more than one route to a given destination, such as work, and choose between them at random, in order to have some variety. In another variant, the drivers preferred route may be closed for construction, forcing him to take a different route.

All of this points to the fact that the prediction of driver intent must be probabilistic. One can predict intent with a certain probability and confidence, but can never be 100% sure of the prediction. For that reason, our approach is to use a statistical model for making the predictions and to learn the model from past driving experience. The basic idea is to collect data on every trip a driver takes and to incrementally

This work was supported by General Motors.

Reid Simmons and Brett Browning are with the Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15224 {reids,brettb}@cs.cmu.edu. Yilu Zhang and Varsha Sedakar are with the Electrical & Controls Integration Lab, General Motors R&D and Planning, 30500 Mound Road, Warren, MI, 48090 {yilu.zhang,varsha.k.sadekar}@gm.com

update the statistical model based on that experience. The model is then used, on-line, to predict driver intent for the next trip, after which that trip is used to, once again, update the model.

The statistical model used in this project is a Hidden Markov Model [1]. A Hidden Markov Model (HMM) is a Markov chain that can be used to track processes with hidden state. In this case, the process is the sequence of driver actions used to follow a route, and the hidden state is the driver's intended destination and route.

While learning HMMs, in general, requires large amounts of data, it is feasible in this case mainly because the structure of the model is already known, it is the road-map of the area, and there are only a small number of transitions between nodes in the model (i.e., a limited number of road segments intersect at any one point). In addition, although the number of road segments in a map database can be very large, the statistical model will be relatively small because any single user tends to traverse only a very small percentage of the roads in a region. Clearly this would not be true for salespeople, delivery people, etc, but in those cases the basic assumption about most routes being routine is also violated, making this approach less applicable for those types of drivers.

III. HIDDEN MARKOV MODELS

A Markov model is a graphical statistical model that captures a sequential model of behavior. It is a tuple $< S, A, T >$, where S is a (finite) set of states, A is a (finite) set of actions, and T is the transition function $T : S \times A \times S \rightarrow \Re$, where $T(s_i, a, s_j) = p(s_i|s_j, a)$, which is the probability of transitioning to state s_i given that the system is in state s_j and action a is executed.

Given a Markov model and an initial state distribution π, one can predict the state distribution that results from carrying out a sequence of actions $< a^1, a^2, ..., a^n >$. If $p^t(s_i)$ is the probability of being in state s_i at time t (where $\pi(.) = p^0(.)$ is the initial state distribution), then $p^{t+1}(s_i) = \sum_{s_j \in S} p^t(s_j) T(s_i, a^{t+1}, s_j)$. Note that while the exact state of the system is uncertain when doing *prediction*, it is assumed that the state is known for certain after the actions are actually *executed* in the world.

A Hidden Markov model (HMM) is a Markov model with hidden (unobservable) state [1]. An HMM is a five-tuple $< S, A, O, T, Z, \pi >$, where S, A, and T are the same as with the Markov model and π is the initial state distribution. In addition, O is a (finite) set of observations and Z is the observation function $Z : O \times S \times A \rightarrow \Re$, where $Z(o, s, a) = p(o|s, a)$, which is the probability of receiving observation o given that the system ends up in state s after executing action a. For many problems, Z is the same for all values of a, (i.e., $Z(o, s_i, a_j) = Z(o, s_i, a_k)$). In what follows, we will use $Z(o, s)$ as shorthand for $Z(o, s, a)$, when Z is the same for all values of a.

As in a Markov model, the exact state of the system is uncertain when predicting the effects of actions. Unlike a Markov model, however, the state may remain uncertain even after executing the actions. For a HMM, both transitions and observations are used to help infer the next state distribution. The same equation as given above for Markov models is used to predict the state distribution at time $t+1$ given the state distribution at time t and the action a^{t+1}. In addition, o^{t+1}, the observation at time $t+1$ is used to further constrain the state distribution:

$$p^{t+1}(s) \leftarrow \frac{p^{t+1}(s) Z(o^{t+1}, s, a^{t+1})}{p(o^{t+1})} \quad (1)$$

Where $p(o^{t+1})$ is a normalizing factor and is given by

$$p(o^{t+1}) = \sum_{s_i \in S} p^{t+1}(s_i) Z(o^{t+1}, s_i, a^{t+1}) \quad (2)$$

IV. PREDICTING DRIVER INTENT WITH A HMM

We now describe our approach to predicting driver intent using a HMM, starting with the road graph representation that underlies the approach. We then describe our basic model containing route and destination information, and the extended model that incorporates contextual information such as time of day, day of week, and vehicle speed.

A. The Road Graph

The core of the representation used in our approach resides in the use of the road graph provided by the mapping database. The mapping database enables us to abstract away from pure GPS locations, towards a street-level graph representation. Fig. 1 shows an example street map using a GUI debugging tool. Concretely, the map is represented as an undirected graph $G = (V, E)$ consisting of vertices V for each intersection, and edges E linking each intersection (i.e. $e = (v_1, v_2)$). Note that each edge in the map database typically contains other information, however, we ignore this for now. We will generally consider a directed version of this graph where edges are ordered to reflect the traveling direction between the intersections. The map database provide a unique one-to-one mapping from each edge to a unique label. We will use the term *link* to indicate the unique labeling of a directed edge between two intersections.

B. The Basic Model

The basic model represents state using pairs $s =< l, g >$, where l is a link and g is a goal (destination). Actions are not represented explicitly, so the transition function T is a function from states to states: $T(s_i, s_j) = p(s_i|s_j)$. Observations are the current link location (a segment in the map database), derived from the vehicles GPS position, speed, and heading. In this work, the observation function is deterministic: $Z(o_l, s) = p(o_l| < l, g >) = 1$, where o_l is the observation of link l. That is, our model assumes that the mapping from GPS position to map link is perfect. In practice, this is not completely true and some pre-filtering is required to remove matching errors. However, it is a reasonable approximation and makes the model considerably more compact, requires less data to obtain a reasonable model, and improves computational performance of prediction algorithms.

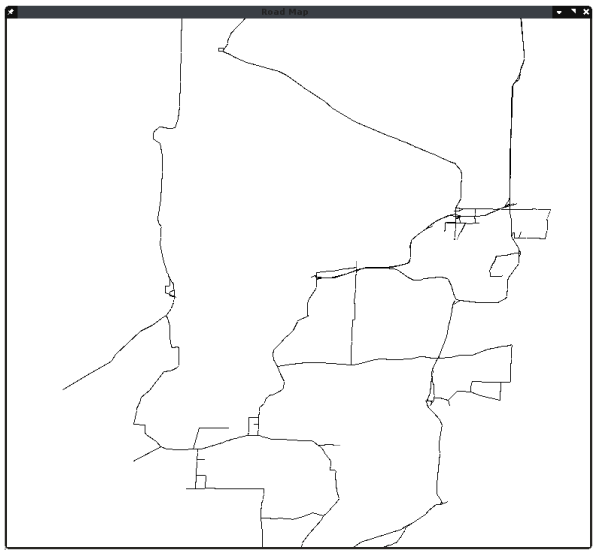

Fig. 1. An example road graph showing the links describing the road segments for a small portion of Detroit.

To reduce the size of the models, the transition probability function is split into two. Instead of $p(s_i|s_j)$, we use $p(l_i|s_j)$ and $p(g|l)$. So, $p(< l_i, g_i > |s_j)$ is determined as $p(g_i|l_i)p(l_i|s_j)$. That is, given the current state we first predict the next link that the driver will go to, and then predict his goal destination, based on that link.

A link l is just an alphanumeric string. In the map database, links are bidirectional. However, the model needs to distinguish which direction the link is traversed. So, each link name from the map data base is appended with an R or L, indicating whether the link is traversed right-to-left or left-to-right. Goals are the same as links (i.e., the link on which the trip ended). Both $p(g|l)$ and $p(l|s)$ are implemented as hash tables. For $p(g|l)$, the key to the hash table is the goal g followed by the link l and the value of an entry is the number of times that l has been traversed when the destination of the trip was g. Another hash table, whose key is the goal links, that stores the number of times that goal has been the destination of a trip.

For $p(l_i| < l_j, g_j >)$, the hash key is the preceding link l_j and the entries of the hash table are vectors of the form $< l_i, g_j, m >$, indicating that the vehicle drove from link l_j to link l_i m times when the goal destination was g_j. The probability $p(l_i| < l_j, g_j >)$ is calculated by dividing m by the sum of all vector entries: $\sum_x < l_x, g_j, m_j >$ (that is, all the transitions from $< l_j, g_j >$). In addition, with each hash entry l_j is stored the mean and variance of the time spent on the link for all trips that traversed that link. Initially, we were planning on using that information to reason about the traversal times (e.g., to predict how long a trip would take, or to predict when a trip was taking significantly longer than average), but to date this data has not been utilized.

The main use of the model is to predict the next link to which the driver will transition. Given a link and a distribution of possible goals, the next link is predicted using $p(l|s)$ and the goal distribution is updated using $p(g|l)$. In particular, we can compute $p(l|s) = \sum p(l| < l_x, g >)p(g)$ from current link l_x, for each known goal.

Given the ability to predict the next link, we can predict a complete route in several ways. One, we can use the goal probability distribution to predict each link in turn, until we reach some goal. Second, we can use the goal probability distribution to predict each link in turn, until we reach the most probable goal. Note that since the prediction of links is used to determine the goal probability distribution, this actually amounts to finding the most likely route through the map, irrespective of the goal. Another approach is to provide the system with a goal, a priori (i.e., $p(g) = 1$) and then use that goal to bias the prediction of subsequent links (ending the route generation when that goal has been reached). In our implementation, each of these methods was available.

C. The Extended Model

For prediction intended goal and route, the basic model is proficient, however, additional accuracy can be achieved with better context. Our extended model incorporates context by augmenting the state representation to with additional factors, such as time-of-day and day-of-week. State is represented as a tuple $< l, g, f_1, f_2, ... f_n >$, where l is a link, g is a goal, and f_i is one of the additional factors. The transition function $T(s_i, s_j)$ and the observation function $Z(o_l, s)$ are defined as in the basic model.

Many of the additional factors (such as time-of-day) are continuous quantities. Since it is difficult to learn HMMs with continuous state parameters, we choose to discretize the factors into a finite number of bins. The idea is that the bins would have semantic meaning that would help to improve prediction accuracy. For instance, we might want to bin time-of-day into pre-dawn (up until 8AM), morning rush-hour (8-10AM), late morning (10-noon), early afternoon (noon-4), evening rush-hour (4-7PM), and night time (after 7PM). Each factor can have its own binning defined, in terms of number of bins and where the bin boundaries occur. Bins are numbered starting with zero, and they are closed on the left and open on the right for instance, the time-of-day bin 8-10AM is interpreted as all times greater-than-or-equal to 8AM and less-than 10AM. The set of bins is circular (this is important for time-of-day and day-of-week), so that any value greater-than-or-equal to the last boundary is considered belonging to the first bin. Thus, in the example given above, the night time bin (after 7PM) would actually be the same as the pre-dawn bin (before 8AM). If one does not want the bins to be circular, one can define a bin whose final value is greater than any legal value. For instance, in defining bins for the speed factor, one can choose the last bin value to be 1000 mph, thus ensuring that no roll-over will occur.

Note that, even for factors that are already discrete (such as day-of-week), it may be desirable to define bins, since this increases the ability to generalize from the data. For instance, one might want to bin day-of-week into weekend and weekday, under the assumption that ones routine trips during the week may differ significantly from those during

the weekend, but that there is not much difference within those two categories. The ability to add new factors and define bins is very flexible, and allows for experimentation to determine which set of bins maximizes prediction accuracy. Note, however, that the HMM model must be re-learned when new factors (or new bins) are added.

Based on this, we define a condition c as a tuple $< g, b_{f_1}, b_{f_2}, ..., b_{f_n} >$, where g is a goal (link) and the b_{f_i} are each binned factors. A state in the extended model can then be represented by $< l, c >$. A condition can be partially specified by having some of its elements be don't care values. Partially specifying conditions can useful when trying to predict the next link, since in certain situations one may not want to completely specify all the factors. For instance, if there is no data in the model corresponding to the current time-of-day, one may want to get a prediction that is independent of the time of day. We say two conditions match if each pair of elements is either equal or at least one is don't care: $match(c_i, c_j) = (g_i = g_j \wedge DontCare(g_i) \wedge DontCare(g_j)) \vee \forall k \in [1,n]\,(b_{f_{ki}} = b_{f_{kj}} \wedge DontCare(b_{f_{ki}}) \wedge DontCare(b_{f_{kj}})$.

For the extended model, a single hash table is used to represent the probability distribution $p(l_i|s_j)$, where s_j is a tuple $< l_j, c_j >$. The hash key is the preceding link l_j and the entries of the hash table are vectors of the form $< l_i, c_j, m >$, indicating that when the condition was c_j the vehicle drove m times from link l_j to link l_i. A special link END_OF_TRIP is used to indicate the situation where link l_j is the last link of the trip.

The transition probability $p(l_i| < l_j, c_j >)$, where c_j may be only partially specified, is determined essentially by marginalizing out the don't care elements of the condition. Specifically, $p(l_i| < l_j, c_j >)$ is calculated by accessing the entry in the hash table associated with link l_j and iterating through the vector elements $< l_k, c_k, m_k >$ to calculate

$$p(m_k|l_i = l_k) = \frac{\sum m_k | l_i = l_k \vee match(c_j, c_k)}{\sum m_k | match(c_j, c_k)} \qquad (3)$$

V. LEARNING THE MODEL

Both the basic and extended HMMs are learned from a set of trips. A trip is a sequence of data points, where each data point includes a link and a time stamp indicating when the vehicle was on the link (note that the same link may appear multiple times in succession, if multiple data points are collected while the vehicle stays on the link). The link associated with the last data point of the trip is considered to be the goal destination of that trip. In addition, for the extended model, each trip data point includes a fully specified condition (a set of binned values representing the additional factors).

The actual learning algorithm is quite simple. The basic idea is to go through the trip sequence and, whenever there is a transition from one link l_j to another l_i, to access element l_j in the hash table and increment the number of transitions to l_i, under the appropriate conditions. More specifically, if the goal link (last link in the trip sequence) is g then, for the basic model, one finds whether the entry $< l_i, g, m >$ exists and, if so, increments m; otherwise a new element is added $< l_i, g, 1 >$. For the extended model, the update is essentially the same, except one is looking for entry $< l_i, c, m >$, where c is the condition of the last trip data point associated with l_j, augmented with the goal link g. Similarly, the number of times the goal g was visited is incremented and, for the basic model, the number of times the pair $< l_j, g >$ was seen is incremented. For the extended model, a transition is added from g to the END_OF_TRIP link. In addition, the statistics for the time spent on l_j are updated, where the time spent on l_j is determined as the time stamp the first trip data point that is on l_j until the time stamp of the first trip data point that is on l_i.

An important assumption for learning models that can yield accurate predications is that the training data is reliable. We have found, however, that the mapping from GPS data to link name is not always accurate (likely due to noise in the GPS data). To improve the fidelity of the learned models, we have developed algorithms to detect and fix two common classes of mapping problems, which we call *jitter* and *stubs*.

Jitter is a phenomenon where the sequence of links in a trip jumps from some main link to another link and then back to the main link, where the two occurrences of the main link are heading in the same direction (see fig. 2 (a) and (b)). The combination of noise in the GPS data and a link close by, roughly parallel to, the main link can cause the wrong link to be chosen. This introduces cycles in the model, which can lead to serious problems in terms of prediction. In particular, the route determination algorithm can loop indefinitely. For that reason, the actual algorithm checks for cycles in the route (i.e., looking for the same link to appear more than once) and terminates if a cycle is found.

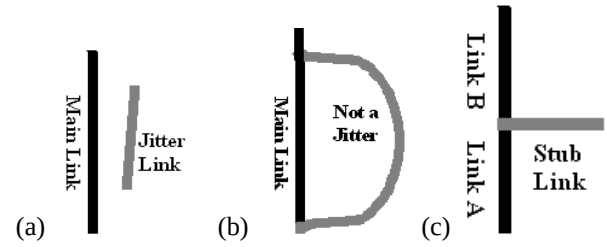

Fig. 2. Schematics showing the types of links that are considered to be jitter noise (a), not true noise (b), and a stub link (c).

Detecting jitter links is relatively straightforward. The algorithm goes through the trip keeping track of the current, last, and penultimate links encountered (note, since the same link may legitimately appear several times in a row, the algorithm updates the current, last, and penultimate link only when there is a change). If the current link is the same as the penultimate link, then the last link is considered to be a jitter link, and is removed from the trip sequence. While this algorithm works correctly in all the data that we have seen, so far, fig. 2 (b) shows a hypothetical situation where the algorithm would incorrectly prune a link.

A stub is a phenomenon where the sequence of links in a trip forms a three-way intersection (see fig. 2 (c)), where the sequence of links in the trip is link A, followed by the

stub link, followed by link B. Stubs arise due to GPS noise, when a link that is perpendicular to, and intersects with, the actual route of the trip is chosen. Detecting stub links is more difficult than detecting jitters. Our current algorithm computes which end of the stub link is closest to link A and which end is closest to link B. If the same end is closest to both links, then we consider it to be a stub link and it is removed from the trip sequence. While there are several hypothetical configurations of curved links that can make this heuristic fail, fortunately, to date, we have not seen examples of them in our data.

VI. EXPERIMENTAL VALIDATION

A number of tests were conducted to test the accuracy of the predictive capability of the model. The models were based on data obtained from GM that represented about a month of urban driving with a low-cost GPS unit. The data set used most extensively included 46 separate trips in the Michigan area. A second data set recorded in the Pittsburgh area but not presented here was also used in development with comparable results. Both data sets included the usual GPS problems found in urban domains: drop outs (common in Pittsburgh due to tunnels and bridges), noise, and bias.

One interesting result is that accuracy, overall, is fairly high, often above 98%. The main reason for this is that around 95% of the transitions in the models are forced, meaning that there is only one next link for a given link. For unforced transitions, the accuracy is still quite good (typically 70-80%), but this accounts for only about 5% of the total predictions. In retrospect, this is not too unusual, since people tend to drive fairly long distances along the same road going from place to place passing many intersections, but turning relatively infrequently. Still, the magnitude of the unforced transitions was surprising. In what follows, we provide results on the percentage of forced transitions and the accuracy for unforced and overall.

For the following tests, we used ten-fold cross validation where the models were learned on 90% of the data (42 trips) and then tested using the other 10% of the data (4 trips). This is repeated 10 times, each time holding out a different set of 4 testing data, with the average over all tests reported. We looked at three additional factors day-of-week (binned into weekend and weekday), time-of-day (binned every 2 hours), and speed (binned into less than 10mph and greater than 10mph). Note that since we use cross-validation, there are some links that appear in the test data that are not included in the training data. Since no reasonable prediction can be made for such links, we report those statistics separately.

Specifically, table I shows each cross-validation test, and the average of all of them. *Not in model* indicates the percentage of links that were not in the model, along with the actual number of such links (in parentheses). Forced percentage indicates the percentage of links for which there was exactly one possible transition, not counting the *not in model* links. *No constraints*, *day-of-week*, *time-of-day*, *speed* and *all* indicate the prediction accuracy with that particular combination of additional factors. The top number in each of those cells is the total percentage of all correct predictions, not including not in model links; The parenthesized number is the percentage of unforced predictions that were correct.

First, note that the percentage of links not in the model is rather small. This helps support our assumption that much of driving is routine. Second, note that the percentage of forced transitions is quite high, averaging nearly 95%. This means that in almost all cases the right prediction will be made without any difficulty. Third, note that the overall prediction accuracy is around 99%. Even with a relatively small amount of training data (less than a months worth of driving), performance is nearly perfect. This bodes quite well for using such an approach on an actual vehicle.

On the other hand, the effect of including additional factors is much less clear-cut. First, note that predicting using speed is the top percentage, or tied for the top. This was somewhat surprising, but the likely explanation is that slowing down is a good indication of when a driver will make a turn (although it could also indicate a stop sign or traffic light). Time-of-day, on the other hand, performed rather poorly compared to some of the other constraints. This may be because the binning we chose (every 2 hours) was suboptimal. It would be useful to try different binnings to see which works best. Finally, note that using all the constraints together is the worst choice. This is likely because with all the constraints being used there may be no data available to make a prediction. We recommend, therefore, to use the all the constraints first and, if the confidence is low, to try using individual constraints. Overall, it is possible that the results would change if there were more training data, increasing the confidence in individual predictions.

The results above were with jitters and stubs removed (see Section 3.4). We ran a test to determine the effect on performance of removing jitters and stubs. Each test was run by learning a model with all of the data and then testing against all the data. This was not done using cross validation, so the percentages are generally higher. The results reported are without any additional constraints predicting the next link given just the previous link. With jitters and stubs removed, the accuracy was 97.8%, with the system predicting correctly 141 times out of 197 unforced situations (71.6%). With just jitters removed, the accuracy increased to 98.1%, with 72.7% accuracy for unforced situations (132 out of 172). With just stubs removed, the accuracy was 98.4%, with 73.4% accuracy for unforced situations (105 out of 143). Interestingly, in this case the accuracy for removing both jitters and stubs was the same as for removing only stubs.

VII. RELATED WORK

Prediction of short-term human behavior is a rapidly growing area of research. In the field of intelligent transport systems, most work has focused on predicting or recognizing short behavior [2], or building cognitive models of how humans make these types of decisions [3]. More specifically to this paper, the work of [4], [5] are perhaps most closely related. [4] makes use of a HMM approach incorporating

	Not in Model	Forced Percentage	No Constraints	Day-of-Week	Time-of-Day	Speed	All
1	36.0% (103)	95.1%	98.4% (66.7%)	98.9% (77.8%)	98.9% (77.8%)	98.9% (77.8%)	98.4% (66.7%)
2	17.1% (24)	92.2%	98.3% (77.8%)	96.5% (55.6%)	96.5% (55.6%)	99.1% (88.9%)	96.5% (55.6%)
3	4.6% (9)	93.1%	99.5% (92.3%)	99.5% (92.3%)	95.7% (38.5%)	99.5% (92.3%)	95.7% (38.5%)
4	16.2% (31)	95.6%	99.4% (85.7%)	100% (100%)	99.4% (85.7%)	100% (100%)	99.4% (85.7%)
5	17.7% (42)	96.9%	99.5% (83.3%)	100% (100%)	100% (100%)	100% (100%)	99.0% (66.7%)
6	7.7% (13)	94.8%	99.4% (87.5%)	99.4% (87.5%)	98.7% (75%)	99.4% (87.5%)	98.7% (75%)
7	7.8% (25)	94.6%	98.3% (68.8%)	98.3% (68.8%)	97.3% (50%)	99.3% (87.5%)	96.3% (31.2%)
8	9.7% (14)	95.4%	98.5% (66.7%)	98.5% (66.7%)	98.5% (66.7%)	98.5% (66.7%)	98.5% (66.7%)
9	15% (59)	94.0%	98.8% (78.9%)	99.1% (84.2%)	99.1% (84.2%)	99.1% (84.2%)	98.8% (78.9%)
10	12.7% (35)	95.8%	98.8% (70%)	98.3% (60%)	99.2% (80%)	99.6% (90%)	99.2% (80%)
Ave.	15.1% (355)	94.8%	98.8% (77.7%)	99.0% (78.6%)	98.4% (68.9%)	99.3% (87.4%)	98.0% (62.1%)

TABLE I

CROSS VALIDATION RESULTS FOR THE PREDICTIVE MODEL AND THE AVERAGE VALUE. RESULTS WERE GATHERED ON A REAL DRIVING DATA FROM A MONTH OF EVERYDAY DRIVING. CATEGORIES ARE: *Not in model* LINKS NOT IN THE MODEL, *Forced percentage* LINKS FOR WHICH THERE WAS ONLY ONE TRANSITION, *No constraints*, *day-of-week*, *time-of-day*, *speed* AND *all* INDICATE THE PREDICTION ACCURACY WITH CONSTRAINT.

a 1D Kalman filter to provide an improved map matching mechanism. [5] uses a HMM of driver routes to also provide better map matching and makes use of the route information for prediction. Our approach differs in that our HMM model focuses on the road link level and utilizes a different representation – each state incorporates knowledge of the destination and also records additional information (time of day, day of week, link duration) that can be used or not during the query process. Our work does not addressed the map matching problem other than to introduce two new simple filtering methods for post-filtering map matches.

In robotics and artificial intelligence, prediction of human behavior has also been explored in a number of settings [6], [7]. For example [8], [9] build probabilistic models of the routes taken by pedestrians using either GPS or an external sensor. These authors used a related graphical model approach (Hierarchical Markov Random Fields). The major difference between their work and ours centers on the underlying representation of state. As explained, our approach exploits the underlying road graph available from the map database and matching mechanisms. In contrast [8], [9] represent the raw position information directly (thus the state is continuous rather than being discrete and sparse). The complexity of the two resulting approaches differ significantly. Given the availability of the map data base and the restricted travel of vehicles to roadways, using a HMM over the road graph is viable for modelling driver routes and leads to significant efficiency gains.

VIII. CONCLUSIONS AND FUTURE WORK

This paper presented an approach to predict driver intent using Hidden Markov Models. The underlying premise is that drivers have certain routine routes and that by learning a model based on previous experience, one can accurately predict what a driver will do in the future. We presented this approach along with experimental results drawn from a corpus of real driving data. The results demonstrate an accuracy of over 98% accuracy of prediction in most cases. Much of this accuracy is due to the fact that there tend to be very few places where choices have to be made most of the predictions are forced. For places where choices are available, the use of additional factors (time-of-day, speed, day-of-week) often improve prediction accuracy immeasurably.

Our future work will focus on extending the model to provide confidence estimates for each prediction, and using this to automate the selection of constraints to provide the most accurate route prediction.

IX. ACKNOWLEDGMENTS

We would like to thank our partners at General Motors Dick Johnson and Sarmad Hermiz for their contributions to this project. Additionally, we would like to thank Carnegie Mellon students Chris Yeung, Dewey Yang, and Ashwin Gupta for their contributions to this project.

REFERENCES

[1] L. Rabiner and B. Juang, *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[2] J. Mccall, D. Wipf, M. Trivedi, and B. Rao, "Lane change intent analysis using robust operators and sparse bayesian learning," in *IEEE International Workshop on Machine Vision for Intelligent Vehicles*, 2005.

[3] D. D. Salvucci, "Inferring driver intent: A case study in lane-change detection," in *Human Factors and Ergonomics Society 48th Annual Meeting*, 2004.

[4] P. Lamb and S. Thiebaux, "Avoiding explicit map-matching in vehicle location," in *The 6th ITS World Congress (ITS-99)*, 1999.

[5] J. Letchner, J. Krumm, and E. Horvitz, "Trip router with individualized preferences (trip): Incorporating personalization into route planning," in *Eighteenth Conference on Innovative Applications of Artificial Intelligence (IAAI-06)*, 2006.

[6] A. Pentland and A. Liu, "Modeling and prediction of human behavior," *Neural Computation*, vol. 11, pp. 229–242, 1999.

[7] D. Schulz, W. Burgard, D. Fox, and A. Cremers, "People tracking with mobile robots using sample-based joint probabilistic data association filters," *International Journal of Robotics Research*, vol. 22, no. 2, 2003.

[8] L. Liao, D. Fox, and H. Kautz, "Hierarchical conditional random fields for gps-based activity recognition," in *International Symposium of Robotics Research*, 2005.

[9] ——, "Location-based activity recognition," in *Advances in Neural Information Processing Systems 19*, 2005.