Perception and Modeling of Scenarios in Ambient Automation Networks with Hidden Markov Models

Dietmar Bruckner
Vienna University of Technology
bruckner@ict.tuwien.ac.at

Josef Mitterbauer
Vienna University of Technology
mitterbauer@ict.tuwien.ac.at

Rosemarie Velik
Tecnalia - Fatronik
rvelik@fatronik.com

Abstract - This paper presents and summarizes methods for machine perception, specifically scenario recognition. Hidden Markov models are used as a mathematical framework. There emission and transition probability distributions are created and updated according to the application-specific algorithms. Tests have been conducted in a senior's home and at a kitchen test bed at the Institute of Computer Technology. Results show that the model is capable of distinguishing various scenarios of the users and that it allows for prediction of future behavior.

Keywords - Building Automation, Machine Perception, Scenario Recognition, Behavior Prediction, Hidden Markov Model.

1. Introduction

Groundbreaking technologies in automation strongly rely on a better embedding in and understanding of the environment. Robots and other machines need to perceive what is going on around them in order to let them act more autonomously. Machine perception of that quality requires a number of diverse sensory sources that produce myriads of data to be pre-processed, fused, interpreted, etc. Standard control algorithms are very much applicable for the first tasks; however they often provide poor results in the latter, in interpreting data.

Data driven methods and black-box learning are often utilized for decision-making units. Due to their very nature they are however unsuited for further interpretation. For proper interpretation it is necessary to provide a model structure where parts of the model can be assigned to "stand for" some particular aspect of the behavior under investigation [1-3].

2. Approach

The presented approach uses hidden Markov models [4] as model framework together with other statistical models [5] for the various probability distributions required. The final model contains representations of behavior in the form of scenarios. Each scenario is constructed from a chain of sensor values. Those values are compared in different ways if they could be represented together or if they need to get their own state. The algorithms for value and state merging are major responsible, if the model later on contains a high or low number of states – which is important for readability [6].

The final model can be used to compare incoming chains of sensor values with already seen and abstracted ones to classify the observed behavior. If a sufficient part of the scenario remains open after classification, the rest of the model data can be used for prediction of future behavior.

3. Hidden Markov Model

There exists a variety of models for modeling a data source which is believed to obey the Markov property - that new values, discrete or continuous, somehow depend on old values. In most cases a first order relationship - that the current value depends somehow on the last one - is assumed.

HMMs in particular are used where it is not possible or useful to directly model observation sequences, but rather to model the underlying source for the change in observations. The Hidden Markov Model consists of (compare fig. 1):

- A list of states $\{Q\}$, $Q_t = 1 \ldots N$, $t \ldots$ time
- A transition matrix $A = \{T_{ij}\}$
- a list of output symbols $\{O\}$, $O_t = 1 \ldots K$
- a confusion matrix $B = \{b_{ik}\}$
- an initial distribution vector π

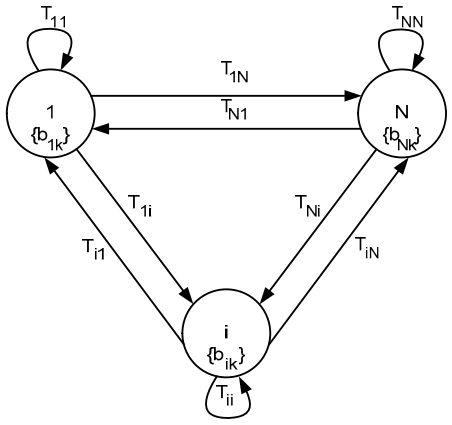

Fig. 1. Scheme of a HMM. It shows 3 of N states, which are connected via transition probabilities. Each state possesses a distribution over output symbols.

4. Structure Learning of HMMs

One approach for choosing the HMM topology is doing it by hand, however, this would be a rather time demanding activity and the optimization process would be very costly. Another technique for choosing structures of HMMs by learning could be to define the number of desired states and using the Baum-Welsh algorithm to learn the connections. This approach is used very often. However, it has the disadvantage that is requires large amounts of training data. Our approach is somewhat different. We translate actual measured chains of values 1 : 1 into state chains that later represent parts of an HMM. Then similarities between such chains are sought in order to merge them and reduce the number of states. This approach requires less training data since actual knowledge about the process is used for creating a first model as opposed to randomly initializing it.

Hence, at the beginning each state of the HMM is associated with exact one value. Due to generalization by state merging, states become a more abstract representation.

State Merging Principles

Two similar or even identical states can be merged into one new state reducing the model's size. The criteria when states can be merged is not part of this section, this is explained later.

Consider two states, I1 and I2 that should be merged into the new state I. Therefore, other states' probability distributions have to be adopted as well. The distributions for transitions and emissions of the new state I are a weighted average of the ones from I1 and I2. Transition probabilities into I are the summed up probabilities into I1 and I2, i.e. the transitions into I1 and I2 have to be redirected to I. Transitions and emissions from I1, I2 come from I after merging. When summing up transition and emission probabilities coming from the new state, weighted averages have to be computed, whereby the weight is the number of visitations of the original states during model creation.

There are two special cases to be considered: Self transitions and transitions between the two states which are to be merged. Therefore, the new self transition probability of the new state after merging has to be recalculated. The states I1, I2 and all probabilities concerning them have to be removed from the model after merging.

Training Data

The higher the number of states in a model, the more specific the underlying system is depicted, while models with a low number of states represent a generalization of the underlying process. The resulting lower accuracy of the latter (in representing the original data) is intended, because it raises the chance that a later perceived scenario can be recognized by the model. However, a too generic model on the other hand is not capable of recognizing differences between scenarios. Hence, one of the key challenges is to find the balance between a very specific model with the risk of not finding a representation for a particular scenario on the one hand and a very generic model, which produces results for every situation but little significance, on the other hand.

In the following, a number of merging heuristics for the application of modeling a persons (or robots) behavior in a room is presented. The input for the model is the position of the robot.

When a robot moves around in the room several sensors are triggered. For each sensor value which is retrieved a new state is created which has exact one emission. The symbol of this emission contains a representation of the sensor value. A new transition is added to the predecessor state with the actually created state as destination. Thus, we get a chain of states where each one has exact one emission, the according position.

Such a chain represents a (very specific) scenario. To get a reasonable model out of a number of those scenarios, we introduce an artificial start state which is the same for all chains, the same with an artificial end state, respectively.

Afterwards the following three steps are applied:

1. Merge Horizontally
2. Merge Vertically
3. Merge Sequences of States

These steps are explained in the following.

Merge Horizontally

The position of a robot is, due to the resolution of the sensors, not an exact value. Sensors that come into question for this application are cameras, pressure sensors on the ground, self-localizing sensors on the robot, etc. However, accuracy in the range of the robot's size or even a bit larger would be enough for a model representation. For this reason the HMM is generalized by merging consecutive states with similar positions into one state that now represents some small area of the room. This step of merging could also be accomplished by some other kind of pre-processing step that abstracts from the actual sensor value to a small area. This step is very necessary for comparing scenarios.

Merge Vertically

To construct a general model of the values learned from all trajectories of the robot, the gained chains are joined together. Only chains with a minimal number of states are added to the global model. The figure below shows such a global model consisting of chains of states. For convenience an artificial start and end state are added. The ellipse around the successors of the start state indicate the first step of merging, the comparison of these states. The gray, dotted chains and states indicate that there are several other chains and states not of relevance for this step of merging.

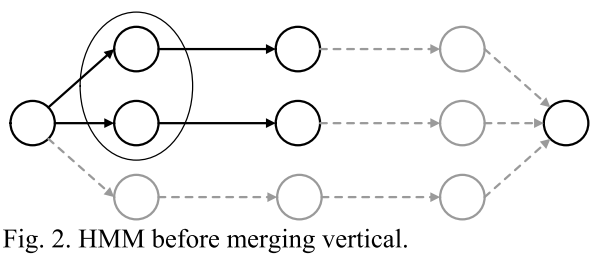

Fig. 2. HMM before merging vertical.

Once the chains are joined together we can merge the model globally. The idea is to reuse parts of chains. Consider the following example, shown in the next figure. The robot moves along a path, i.e. it produces the sequence of positions (A, B, C, D, E). Another trajectory produces the sequence (A, B, C, F, G). In our model this is represented by two chains, corresponding to the positions. These chains are similar until the point where one trajectory proceeds to (D) and the other to (F). So we can merge the start of the sequences in our model and stop where the trajectories differ.

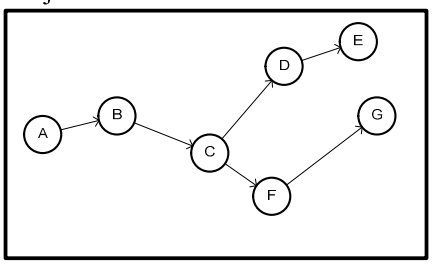

Fig. 3. Splitting Paths of two different trajectories.

For our algorithm this means: beginning from the artificial start state, any two successor states of any state are compared if they are similar (within an application dependent similarity criterion). If similarity is given, the states are merged. This procedure is repeated until nothing could be merged anymore. In the example above, the first three states could be merged, because the sequence (A, B, C) is the same for both trajectories. A hypothetical resulting model of Vertical Merging from the start is shown in Figure 4. The same can be done with the end of the sequences. So we use the same algorithm starting from the artificial end state in back direction.

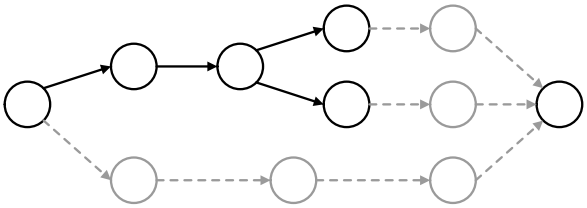

Fig. 4. HMM after merging vertical.

Merge Sequences

As described in the previous section we get a model where similar start-sequences and end-sequences are merged. The limitation is that only two states are compared. An enhanced version of this approach is to search the model for sequences of several states in the midst of the model, like shown in figure 5. If two similar sequences are found, the start and the end state of such a sequence are merged. The states between start and end of one chain become useless since they cannot be reached anymore. However, this algorithm can cause back transitions. As can be seen the same sequence can theoretically appear several times within one chain. If the algorithm is applied several times, all occurrences of the sequence (A, B, C) are merged together. This results in a model as shown in figure 6. We get a transition from a state later in the scenario back to a state which occurred earlier. This transition is called a back transition. These transitions violate the temporal order of states in a scenario.

Back transitions could be suppressed, but it might be of interest allowing them. If it is useful to allow back transitions is dependent on the application. E.g. if the robot's motion model is learned for fulfilling tasks that require multiple movements in the same area, the concept of start and end states is reduced to a mathematical need, and in such a case the interesting trajectories are anywhere in the model whereby the intrinsic correlation between temporal order and position in the model becomes useless.

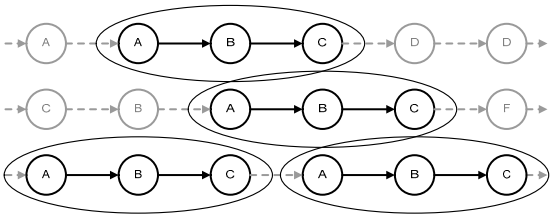

Fig. 5. Comparing and merging sequences of states.

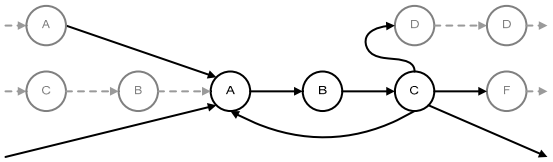

Fig. 6. Back transition as a consequence of sequence merging. Additional to figure 5, altered transitions are drawn black.

4. Resulting Models

In this section, a model, which is learned from motion detector data in a senior's home, is presented. Every path through the model represents a particular daily routine. Hence, the model observes human behavior in one particular room. In this context we can talk of a semantic concept for a whole day of that area. But, moreover, some of the states themselves also represent particular - by humans identifiable - parts of a daily routine. Examples for states could be "lunch", "time before getting up", or "bathroom visit after lunch" and for (sub) paths something like "afternoon with low activity". Fig. 7 is to be read in the following way:

- The circles represent the states of the model. They are labeled with numbers from 0 to N, whereby state 0 is the "initial state" which is part of every chain and state N is the "final state".
- The lines represent possible transitions. Only the drawn transitions are nonzero and learned during the above mentioned procedures.
- Self-transitions have been omitted for readability since all states have them.

Figure 8 shows one particular day that was used to create the model in figure 7. In sum, 15 days have been used to create the model. The log-likelihood of -20,5 translates to an average transition and emission probability of 80,8%. In the whole analysis of the data from the elderly home [1] the best paths have probabilities in the range of 73% to

83% whereas the worst paths (which are also possible for a chain of values, but with lowest probabilities) are between 62% and 78% (note: some paths are not applicable for some data and therefore have still zero probability). These figures give an impression about the generalization during merging. Without merging, that data in figure 8 would have been recognized with 100%. However, any other data would have zero probability. After merging, a perfect fitting scenario reaches up to 83% of accuracy while any other scenario that is similar to it also gets reasonable values.

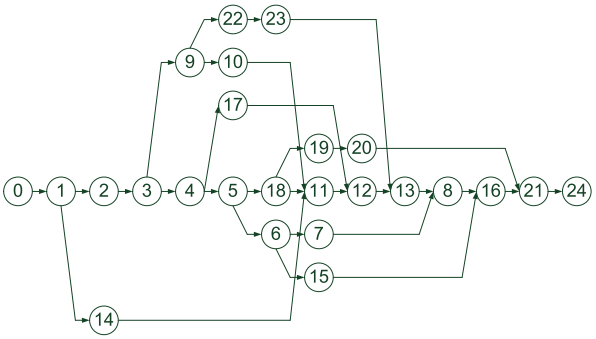

Fig. 7. Model of a motion detector sensor located in the bathroom; 23 states.

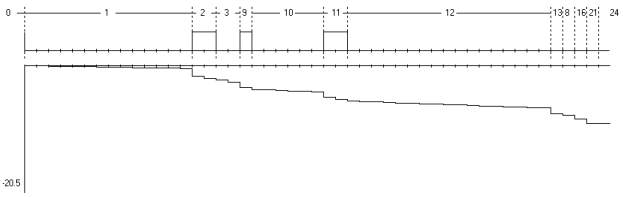

Fig. 8. Sensor values and Viterbi path of one actual day in the model from figure 7. The bottom part shows the progression of the log-likelihood.

5. Prediction

Prediction is computed for positions of persons or robots learned from pressure sensors in the kitchen environment of the institute. The same procedure could be applied for the data from the elderly home. However, this data set is chosen for compliance with the conference topic.

It is conducted in the following way: first of all, a distribution over states in the model is calculated that fits to the previous sequence of sensor values. Then, all possible next states are determined and a weighted average is calculated. This average represents the predicted next position.

The quality of prediction is analyzed by comparing the predicted position to the position which occurred in reality at the next step. We define the distance between these positions as the quality of the prediction. The test scenario is quite simple: A person enters the empty room (i.e. there are no other persons), walks to the fridge, opens the fridge, closes the fridge, goes on to some other position, turns around and leaves the room. Fig. 9 shows the evaluation of an HMM that learned trajectories in that room. At the x-axis there is the count of steps, at the y-axis the distance as mentioned above. The series d(Pi) represent the prediction with the i^{th}-best accuracy. As can be seen, d(P1) is almost always in the range of 0.5 m. d(P2) has some higher deviations and d(P3) has still more high deviations than d(P2). Most of these deviations are due to the fact that the HMM got stuck with a scenario and did not have alternatives. In this case the position of the door was the next estimate …

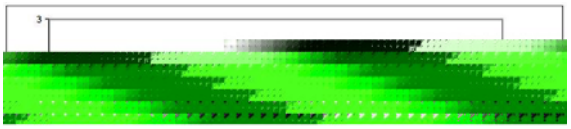

Fig. 9. Evaluation of prediction in the kitchen environment.

6. Conclusion

It can be shown that the proposed hidden Markov model approach works well for modeling behavior, classifying behavior, and even predicting future behavior. Models have been constructed that represent the daily routine of persons living in a senior's home. Particular states can be interpreted as particular parts of the person's schedule.

Furthermore, the model can also be used for prediction. It works for daily routines as well but was only shown here with a model representing positions of moving persons/robots. The accuracy of prediction is mostly within the range of one step.

References

[1] D. Bruckner, B. Sallans, and R. Lang: "Behavior Learning via State Chains from Motion Detector Sensors", Proceedings of IEEE BIONETICS, 2007.

[2] J. Mitterbauer, D. Bruckner, and R. Velik: "Behavior Recognition and Prediction in Smart Building Automation Systems", Proc. of IFAC FET 2009.

[3] D. Bruckner: "Probabilistic Models in Building Automation: Recognizing Scenarios with Statistical Methods", Dissertation Thesis, Vienna University of Technology, 2007.

[4] L. R. Rabiner and B-H. Juan: "An Introduction to Hidden Markov Models", IEEE ASSAP Magazine, vol. 3, p. 4-16, 1986.

[5] C. M. Bishop: "Neural Networks for Pattern Recognition", Oxford University Press Inc., New York, 1995.

[6] A. Stolcke and S. Omohundro: "Hidden Markov Model Induction by Bayesian Model Merging". In: S. J. Hanson, J. D. Cowan, and C. L. Giles (eds.): "Advances in Neural Information Processing Systems", vol. 5, p. 11-18, Morgan Kaufmann, San Mateo, 1993.