# Parallel Hypothesis Driven Video Content Analysis

Ole-Christoffer Granmo
Agder University College
Grooseveien 36, N-4876 Grimstad, Norway
ole.granmo@hia.no

## ABSTRACT

Extraction of features from images, followed by pattern classification, is a promising approach to automatic video analysis. However, a *parallel processing environment* is typically required for real-time performance. Still, single-CPU Bayesian network systems for *hypothesis driven feature extraction* have been able to classify image content real-time — the expected information value and processing cost of features are measured, and only efficient features are extracted. The goal in this paper is to combine the processing benefits of parallel and hypothesis driven approaches. We use dynamic Bayesian networks to specify video analysis tasks and the particle filter (PF) for approximate inference, i.e., feature selection and classification. The inference accuracy of any given PF is determined by the number of particles it maintains. To increase the number of particles maintained without reducing the processing rate, we apply multiple PFs distributed in a LAN, and a *pooling system* to coordinate their output. Our resulting multi-PF architecture supports three video frame processing phases: a parallelized *feature selection* phase, followed by a parallelized *feature extraction*- and *classification* phase. Unfortunately, we observe a loss of inference accuracy when splitting a single PF into multiple independent PFs. To reduce this loss, we let the pooled PFs exchange particles across the LAN. An object tracking simulation demonstrates the ability of our architecture to select efficient features as well as the effectiveness of our particle exchange scheme — we observe a significant *increase* in inference accuracy compared to the tested non-parallel PF.

## Keywords

Real-time Video Content Analysis, Feature Selection, Particle Filtering, Bayesian Networks, Parallel Processing

## 1. INTRODUCTION

The technical ability to generate volumes of digital video data is becoming increasingly "main stream". To utilize the growing number of video sources, both the ease of use and the computational flexibility of methods for content based access must be addressed.

In order to make video data more accessible, pattern classification systems which automatically classify video data in terms of high-level concepts have been taken into use. The goal of such systems is to bridge the gap between the low-level features produced through signal processing (e.g., color histograms and motion vectors) and the high-level concepts desired by the end-user (e.g., "running person").

In the above context, dynamic Bayesian networks (DBNs) [14] represent a particularly flexible class of pattern classifiers that allows statistical inference and learning to be combined with domain knowledge. Indeed, DBNs have been applied to a wide range of video content analysis problems, e.g., [1,7,9,12,20]. The successful application of DBNs can in many ways be explained by their firm foundation in probability theory, combined with the effective techniques for inference and learning that have been developed.

The particle filter (PF) [13, 16, 17] is an approximate inference technique that opens up for *real-time* DBN-based event detection/tracking in video. That is, the PF supports content-based analysis of video frames as they are captured. A real-time building surveillance system could for instance automatically classify whether someone is running rather than walking, on-line. To elaborate, in [16] the PF is used for probability density propagation that allows simultaneous tracking and verification in video.

A video content analysis application is typically required to process video frames at a certain rate, or to respond to real world events within a certain time limit. In such cases, attention must be paid to the processing environment used as well as to the selection and configuration of feature extraction and classification tasks.

There are two main approaches to meeting processing rate/response time requirements. The first approach is to parallelize the feature extraction and classification to take advantage of multiple CPUs. The processing rate/response time of an application can then be improved by making more CPUs available for processing. For instance, in [8] an extensible and modular software architecture for parallel processing of data streams is proposed. Furthermore, in [18] it is shown how event response time can be improved by distributing the logical processing tasks of a video surveillance application in a physical processing architecture consisting of multiple connected CPUs. Similarly, [4, 10] describe a framework for coarse-grained multi-level parallelization and distribution of video stream filtering/transformation, feature extraction, and classification.


*SAC* '04, March 14-17, 2004, Nicosia, Cyprus

The second approach to meeting processing rate/response time requirements is based on the assumption that feature extraction typically is significantly more processing intensive than classification. Feature extraction may involve costly on-line signal processing while a pattern classifier is normally trained/specified off-line, such that it performs efficiently on-line. Thus, the second approach, referred to as hypothesis driven feature extraction, is based on only extracting selected features. The goal is to minimize feature extraction processing cost, while still maintaining acceptable classification accuracy. E.g., in the Bayesian network based image classification system from [15], features are extracted sequentially in decreasing order of efficiency (value of information relative to cost of processing). When the efficiency falls below a threshold, the feature extraction stops. Likewise, [2] is a recent approach from the field of physical sensor management which supports selection of the most "informative" sensor at each time step of a tracking task.

By integrating the parallel and the hypothesis driven video content analysis approaches, it should be possible to achieve further processing rate/response time improvements. However, several difficulties are introduced. Selected features should be extracted in parallel to support computationally costly signal processing. Also, the actual *feature selection*, as well as the *classification*, should be parallelized to support complex, accurate, and/or timely video content analysis.

In this paper we propose a logical video content analysis architecture that targets the above described difficulties. As a basis, we overview DBNs and PFs in Section 2. In Section 3, we show how the so-called *pooled* classifiers architecture [3] can be applied to execute multiple PFs in parallel in a Local Area Network (LAN). We also propose a three-phase procedure for processing video frames, consisting of parallel feature selection, feature extraction, and classification. To reduce the loss of inference accuracy caused by splitting a single PF into multiple independent PFs, we then suggest a communication scheme that allows PFs to exchange particles across a LAN. The resulting techniques are evaluated empirically in Section 4. We conclude in Section 5.

## 2. DYNAMIC BAYESIAN NETWORKS AND PARTICLE FILTERING

We use DBNs to specify video content analysis tasks, and we shall here give a short introduction. For a more thorough treatment, readers are referred to e.g. [14,21]. We also review the principles behind PF based inference.

### 2.1 Dynamic Bayesian Networks (DBNs)

A DBN consists of a qualitative and a quantitative part. The qualitative part is a directed acyclic graph (DAG). The *nodes* in the DAG are variables with states. Each variable represent an entity of interest, and the states of a variable represent the states the corresponding entity can be in. For instance, consider a video frame divided by an $8 \times 8$ grid. A motion feature is extracted from each grid position, and the goal is to determine the coordinates of a tracked object from these motion features. Then, one DBN variable could represent the position of the object. This variable would have 64 states, one for each grid position. Furthermore, we could assign a motion feature variable to each grid position with states "low", "medium", and "high". These variables would represent the output of the motion feature extractors.

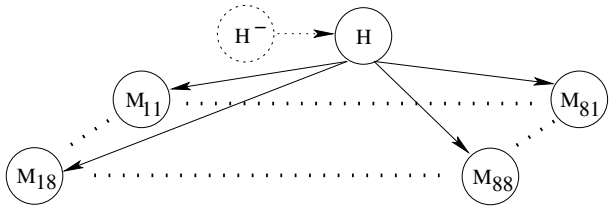


**Figure 1: A DBN modeling object tracking.**

The *directed links* in the DAG represent causal impact between the variables. E.g., a moving object influences the features output by the motion feature extractors, and therefore we should add a link from the object position variable to each motion feature variable. So far, the above description corresponds to an ordinary Bayesian network (BN). The difference between a BN and a DBN is that, in a DBN, the DAG is sectioned into a sequence of identical *time slices*. Each time slice represents a particular interval in time. This means that links between variables in two consecutive time slices indicate a causal impact from time interval to time interval. In a DBN, we could for instance let each time slice correspond to a video frame and then relate the possible positions of a moving object from time slice to time slice.

The qualitative DBN part suggested above represents an object tracking task and is illustrated in Figure 1. This DBN consists of $8 \times 8$ motion feature variables $M_{ij}$, organized spatially as a grid, and a hypothesis variable $H$ representing the grid position of a tracked object. The inter time slice link $(H^-, H)$ reflects that the position of the object depends on its position in the previous time slice. Similarly, the link from $H$ to a motion feature $M_{ij}$ reflects the causal impact of each object position on the motion feature at grid position $i, j$. Note that in a stationary model, the causal impact between time slices does not vary. Thus, only the causal impact between two arbitrary consecutive time slices needs to be specified, as illustrated in the figure.

We now turn to the quantitative part of a DBN — the strength of the directed links are represented as conditional probabilities. For each variable $A$ with parents $pa(A)$, we have to specify the conditional probabilities $P(A \mid pa(A))$. If $pa(A) = \emptyset$ we specify the prior probabilities $P(A)$. So, for the DAG in Figure 1 we have to specify $P(H^-)$, $P(H \mid H^-)$, $P(M_{11} \mid H) \cdots P(M_{88} \mid H)$. E.g., we could set $P(M_{11} = \mathit{high} \mid H = [6,6]) = 0.05$ in order to indicate that significant motion in block $[1,1]$ is unlikely when the object is located at grid position $[6,6]$.

The above DBN model is used as a basis for the object tracking application described in [4,5]. Other kinds of content analysis tasks can be modeled in a similar manner. To conclude, we mainly use DBNs to calculate *posterior probabilities*, i.e., to calculate the probability of certain variables being in certain states, given the observed states of other variables. E.g., assume that we consider a single video frame and that the states of the motion feature variables have been observed for that frame: $M_{11} = m_{11}, \ldots, M_{88} = m_{88}$. We could then calculate $P(H = [i,j] \mid M_{11} = m_{11}, \ldots, M_{88} = m_{88})$ for each coordinate $[i,j]$ and identify the a posteriori most probable object position. Note that we are free to select which features to observe, and the selection determines how accurately the position of the object can be decided.

```
1:  % Extend particles to cover time slice t+1 from t
2:  S' := copy(S);
3:  FOR EACH s ∈ S DO
4:   % Replace particle s based on sampling from
5:   % distribution defined by particle weights
6:   s := SAMPLE P(S');
7:   % Assign state to DBN variables in new time slice
8:   FOR EACH X^{t+1} ∈ 𝒳^{t+1} DO
9:    IF X^{t+1} ∈ ℋ^{t+1} THEN
10:     s.X^{t+1} := SAMPLE P(X^{t+1} | pa(X^{t+1}) = s.pa(X^{t+1}));
11:   ELSE
12:     s.X^{t+1} := OBSERVE X^{t+1};
13:     s.w ×= P(X^{t+1} = s.X^{t+1} | pa(X^{t+1}) = s.pa(X^{t+1}));
```

**Figure 2: The updating step of the PF algorithm.**

## 2.2 A DBN Based Particle Filter (PF)

There exist many algorithms for calculation of posterior probabilities in DBNs [14,21]. The PF [13,17] is a *real-time* approximate technique. We here describe the PF in two stages. I.e., we first describe the state of a PF at time slice $t$ (including $t = 0$). From this state, posterior probabilities can be calculated given features observed up to and including time slice $t$. We then describe a procedure for advancing to time slice $t + 1$.

Let $\mathcal{X}^t$ denote the set of DBN variables in time slice $t$. Furthermore, let $\mathcal{H}^t$ denote the so-called hypothesis variables of time slice $t$, that is, the variables $\mathcal{H}^t \subseteq \mathcal{X}^t$ which cannot be observed directly. Finally, let $\mathcal{F}^t$ denote the so-called feature variables in time slice $t$, i.e., the variables $\mathcal{F}^t \subseteq \mathcal{X}^t$ whose states can be observed.

For the current time slice ($t$) our PF maintains a set $S$ of *particles*. The particles can be seen as weighted samples. A single particle $s$ consists of two parts. The first part of $s$ is simply an assignment of a state $x^t$ to each hypothesis variable $X^t \in \mathcal{H}^t$: $s.X^t := x^t$. Similarly, each feature variable $X^t \in \mathcal{F}^t$ is assigned its observed state. In our example, a particle assigns an object position to the current video frame as well as a degree of motion to each grid position.

The second part of $s$ is a weight $s.w$ which determines the probability of the feature observations up to and including time slice $t$, given the states $s$ assigns to the hypothesis variables: $P(\mathcal{F}^1 = f^1, \ldots, \mathcal{F}^t = f^t \mid \mathcal{H}^1 = s.\mathcal{H}^1, \ldots, \mathcal{H}^t = s.\mathcal{H}^t)$. Accordingly, this part indicates how accurately the assignments of $s$ reflect the "truth". Note that each particle is initialized by sampling from a prior distribution $P(\mathcal{X}^0)$ specified as a part of DBNs, e.g., $P(H^-)$ in our example DBN. The particle weights are set to 1 — no observations done for $t = 0$.

By normalizing the weights of the particles, the particles can be seen as an approximation of the joint posterior probability distribution $P(\mathcal{H}^t \mid \mathcal{F}^1 = f^1, \ldots, \mathcal{F}^t = f^t)$ [21]. From this joint probability distribution, the posterior distribution of individual hypothesis variables can be calculated.

The question then is how to update the particle set $S$ so that it covers a new time slice $t + 1$. An algorithm which performs this updating step is found in Figure 2 and detailed below. Each particle $s$ is updated as follows. First, $s$ is replaced by a particle drawn randomly from the particle set. The particle is drawn according to the probability distribution derived by normalizing the particle weights, hereafter denoted $P(S')$. Generally, the resulting replaced particle set will provide a better starting point for approximating $P(\mathcal{H}^{t+1} \mid \mathcal{F}^1 = f^1, \ldots, \mathcal{F}^{t+1} = f^{t+1})$ compared to the original particle set. This is because particles representing less likely scenarios are equivalently less likely to be included in the new particle set. Thus, the particles are concentrated on the likely scenarios and will not spread out across an exponentially large set of possible, but unlikely scenarios.

As a second step, the DBN variables in time slice $t + 1$ are ordered topologically with respect to the DBN DAG and assigned states by particle $s$ in that order. Each hypothesis variables $X^{t+1} \in \mathcal{H}^{t+1}$ is assigned a state drawn from the conditional probability distribution $P(X^{t+1} \mid pa(X^{t+1}) = s.pa(X^{t+1}))$. Because of the ordering of the variables, the particle has already assigned states to the parents $pa(X^{t+1})$ of $X^{t+1}$. The feature variables are treated differently as the states of these are given. In short, the particle weight $s.w$ is updated to include each new observation ($s.X^{t+1} :=$ OBSERVE $X^{t+1}$): $s.w := s.w \times P(X^{t+1} = s.X^{t+1} \mid pa(X^{t+1}) = s.pa(X^{t+1}))$.

In essence, the above mechanisms evolve the particles to be a summarization of likely video frame interpretations.

# 3. A PARALLEL HYPOTHESIS DRIVEN VIDEO ANALYSIS ARCHITECTURE

The limited processing resources available on a typical host restrict the complexity, accuracy, and timeliness of video content analysis tasks. In this section, we apply the *pooled* classifiers architecture [3] to PFs in order to support parallel hypothesis driven video content analysis — multiple PFs execute in parallel and a pooling system coordinates their output. We also propose a PF communication scheme for exchanging particles, with the goal of increasing inference accuracy.

## 3.1 Pooled Classifiers Architecture and PFs

In a traditional pooled classifiers architecture [3], $k$ independent classifiers take a set of features as input. The classifiers are executed in parallel so that $k$ classifications are output. A pooling system aggregates these outputs to make a final classification.

We adopt the principles of the pooled classifiers architecture to PFs. In short, PFs are used as classifiers and a pooling system coordinates their output. The processing of each video frame $t$ is done in three phases — a feature selection phase, a feature extraction phase, followed by a classification phase. We parallelize the three phases as shown in Figure 3. The figure illustrates that the available CPUs switch between extracting features and filtering particles. In between, a pooling system executes on a selected CPU. The CPUs are connected in a LAN, and communication is based on transmitting messages across the LAN. Modern LANs typically allow messages to be transferred between machines in a few microseconds or so [22] — a small amount of time in the context of video content analysis (normally at least 40 milliseconds are available for processing each video frame).

Let us now take a closer look at the above indicated processing phases and the resulting communication. Phase one is illustrated in Figure 4. The goal in phase one is to identify the $n$ features, $\Phi^t \subseteq \mathcal{F}^t$, which are most efficient when it comes to determining the content/events of time slice $t$. To

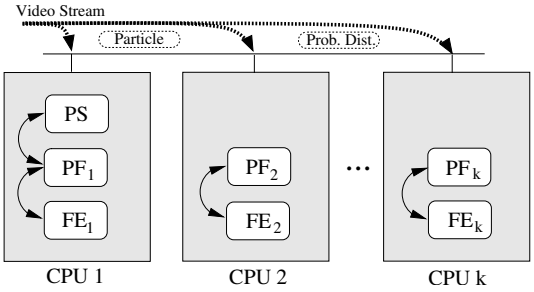


**Figure 3: CPUs in a LAN — each CPU switches between feature extraction (FE) and particle filtering (PF). A Pooling System (PS) executes on a selected CPU. The dotted ovals represent LAN messages.**

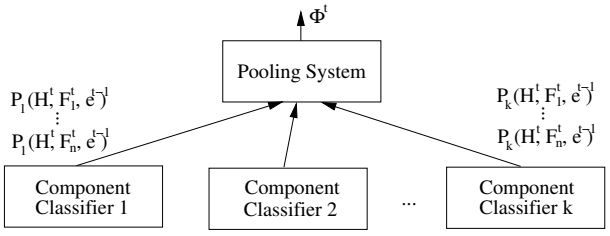


**Figure 4: Feature selection phase.**

elaborate, each PF $i$ updates its particles to cover time slice $t$ in the traditional manner (see Figure 2), however, features have not yet been extracted in time slice $t$ so particle weights are not updated. For each feature variable $F_j^t \in \mathcal{F}^t$, the particles are then used to approximate the joint hypothesis-feature probability distribution $P(\mathcal{H}^t, F_j^t, e^{t-1})$. Here, $e^{t-1}$ denotes features extracted from previous video frames. In essence, $P(\mathcal{H}^t, F_j^t, e^{t-1})$ describes the pairwise interplay of feature variable $F_j^t$ and the hypothesis variables $\mathcal{H}^t$, in light of previous observations $e^{t-1}$. As a last step, each PF submits the resulting set of joint hypothesis-feature probability distributions to the pooling system, as illustrated in Figure 3 and Figure 4.

For each feature $F_j^t \in \mathcal{F}^t$, the pooling system then sums and normalizes the $k$ local approximations of $P(\mathcal{H}^t, F_j^t, e^{t-1})$ it has received from the PFs. By mimicking a non-parallel PF in this manner, a set of joint global posterior hypothesis-feature distributions are produced: $\hat{P}(\mathcal{H}^t, F_j^t | e^{t-1})$, $F_j^t \in \mathcal{F}^t$. These global probability distributions summarize the "a posteriori" interaction between each feature variable $F_j^t$ and the hypothesis variables $\mathcal{H}^t$ and accordingly, form the basis for the simple *entropy* calculations required to determine the efficiency $Ef(F_j^t)$ of each feature variable $F_j^t$:

$$Ef(F_j^t) = \frac{H(\mathcal{H}^t) - H(\mathcal{H}^t \mid F_j^t)}{Cost(F_j^t)}.$$

I.e., the entropy of the hypothesis distribution, $P(\mathcal{H}^t)$, is used to measure the current hypothesis uncertainty:

$$H(\mathcal{H}^t) = -\sum_{h^t \in \mathcal{H}^t} P(h^t) log[P(h^t)],$$

and the *expected entropy* of the hypothesis distribution is used to measure the hypothesis uncertainty to be expected after extracting a feature $F$:

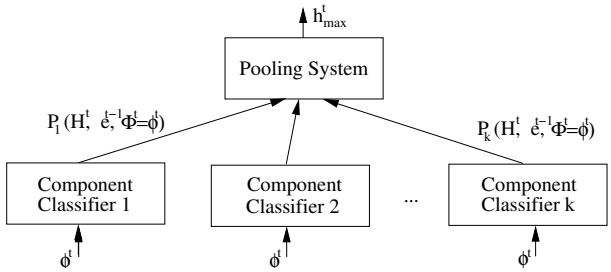


**Figure 5: Classification phase.**

$$H(\mathcal{H}^t|F) = -\sum_{f \in F} P(f) \sum_{h^t \in \mathcal{H}^t} P(h^t|f) log[P(h^t|f)].$$

In other words, the so-called *Mutual Information Gain* divided by the feature extraction processing cost is used to measure the efficiency of each feature in time slice $t$. Finally, the pooling system outputs the $n$ most efficient features $\Phi^t$.

In phase two, the selected features are extracted in *parallel* on the available CPUs, as determined by the pooling system. Note that we assume that the video frames are transmitted to the CPUs by means of so-called multicast as indicated in Figure 3 and discussed further in [5, 6].

Finally, phase three is illustrated in Figure 5. In phase three, each PF $i$ takes the features extracted in the current time slice $t$ as input, i.e., $\Phi^t = \phi^t$ is taken as input. Based on this input, each PF updates its particle weights and outputs an approximation of the unnormalized hypothesis probability distribution $P(\mathcal{H}^t, e^t)$, as follows from Section 2. Note that, $e^t \equiv e^{t-1} \cup \{\Phi^t = \phi^t\}$. The pooling system sums and normalizes the output local approximations, to produce a global posterior hypothesis probability distribution $\hat{P}(\mathcal{H}^t|e^t)$. This global probability distribution is used to identify the posterior most probable hypothesis state $\mathcal{H}^t = h^t_{\max}$, which the pooling system outputs. Accordingly, the content of time slice $t$ is classified. This concludes the processing of time slice $t$.

## 3.2 A Scheme for Exchanging Particles

Unfortunately, the above approach to parallelizing PFs has a significant disadvantage. The approach fragments the particles of the PF to be parallelized into $k$ sets, one for each classifier component. This influences the particle replacement conducted at line 6 of the PF algorithm in Figure 2. To explain the consequences of this fact, let us first consider some of the characteristics of the traditional PF.

The traditional PF is mainly model-driven and not data-driven. That is, only the prior hypothesis state distribution of a new time slice $t$, as modeled by the DBN, is used for sampling (see line 10 from Figure 2). The features extracted in the time slice (the data) are not considered at all when sampling the hypothesis states. So, if the true states of the hypothesis variables are improbable given the prior distribution, few particles will match those states (how few depends on how improbable the true states are). For instance, assume that a DBN models the movement patterns of a certain class of airplanes. If the pilot of an airplane that is tracked by a PF manages to make a difficult and therefore improbable maneuver, few particles will match that move. This may cause problems for the PF. Indeed, if no particles match the true hypothesis states, the PF may lose track of the current

real-world situation. On the other hand, if even a single particle is able to match the true hypothesis states, the particle replacement step of line 6 in Figure 2 makes sure that most of the particles are brought back on track.

However, this particle replacement effect is significantly reduced when the particles are fragmented into small sets as in the above pooled classifiers architecture: *particles are only replaced from the locally available particles*. Consequently, each individual PF in the pool may lose track of the real-world situation one-by-one. As PFs start loosing track of the real-world situation, increasingly amounts of noise are introduced to the pooling system. Consequently, the classification suffers.

To overcome the above fragmentation problem, we now introduce a scheme for exchanging particles between the pooled PFs. The main purpose of exchanging particles is to bring strayed PFs back on track.

The scheme takes advantage of LAN support for broadcasting messages. Note that other classes of efficient one-to-many communication mechanisms may also be used, such as multicast. Broadcasting a particle means that the particle is received by all the pooled PFs by the means of a single LAN transmission. In contrast, traditional unicast communication requires one LAN transmission for each recipient. Accordingly, if each of the $k$ PFs in the pool are to submit a particle to each of the other PFs in the pool, we would need $k(k-1)$ LAN transmissions when applying unicast. By using broadcast, on the other hand, we only need k LAN transmissions for this particular particle exchange. Accordingly, by taking advantage of the broadcast facilities of a LAN, efficient sharing of particles may be achieved.

To elaborate, we add a particle broadcast step to the PF algorithm from Figure 2:

$$\texttt{14:}\quad \texttt{BROADCAST}\ \text{argmax}_{s\in S}\, s.w\texttt{;}$$

In this step, each PF identifies the particle with the largest weight and then broadcasts this particle to the other PFs in the pool. This means that when each PF advances to a new time slice, they have been supplied with $k-1$ additional particles. As a consequence, line 2 of the PF algorithm is modified so that local particles also can be replaced by the supplied particles ($S_B$):

$$\texttt{2:}\quad S' \texttt{ := copy}(S \cup S_{\text{B}})\texttt{;}$$

Note that the particle weights have to be normalized repeatedly in order to avoid the difficulty of representing small real-valued numbers on computers. In our distributed scheme, normalization is supported by piggy-backing normalization constants with the broadcasted messages.

To conclude, the scheme makes sure that each PF is provided with the locally most likely particles in the PF pool, by only transmitting a number of messages equal to the number of PFs across the LAN at each time step. Most importantly, this scheme guarantees that the globally most likely particle in the PF pool always is shared between the PFs.

# 4. EMPIRICAL RESULTS

In this section we evaluate the proposed techniques empirically by the means of the DBN object tracking model from Figure 1. In order to challenge the model-driven PF approach, we simulate an object that makes unexpected moves from time slice to time slice, i.e., the movement does not conform to the tracking model. The tracked object could for instance be an airplane whose pilot tries to confuse the tracker by making unexpected maneuvers. To elaborate, we let the object movement between two time slices have probability 0.04 of occurring a priori, as illustrated in Figure 6. With 128 particles (on track) the probability that no particle matches a given movement is then $0.96^{128} \approx 0.005$, and with 16 particles the probability is $0.96^{16} \approx 0.52$. Although an object can be in any of the 64 grid positions, we see from Figure 6 that it only moves to one of the 8 adjacent grid positions in the transition between two time slices. We also see that the object makes the a priori least probable move of the possible moves. Note that the object is only detectable by the motion sensor at its current location.

| | | |
|---|---|---|
| 0.04 | 0.12 | 0.04 |
| 0.12 | 0.36 | 0.12 |
| 0.04 | 0.12 | 0.04 |

**Figure 6: The modeled probability of each possible object move centered on the current object position.**

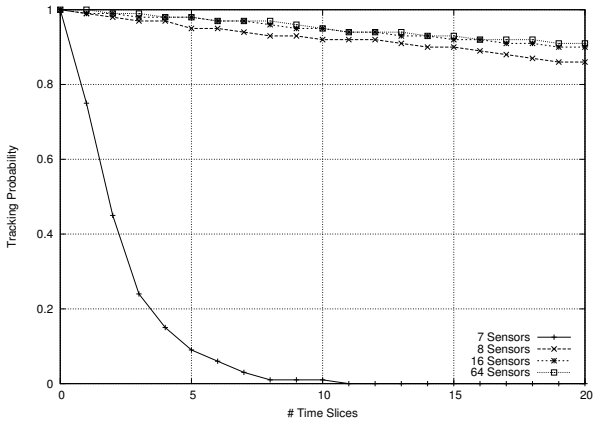


**Figure 7: The tracking probability of 8 parallel PFs exchanging particles, after # time slices.**

The pooled PFs architecture is tested with 8 PFs, each maintaining 16 particles. The architecture is evaluated both with and without the particle exchange scheme. Also, a non-parallel *single* PF that maintains 128 particles is tested for comparison purposes. The number of time slices passed before a technique loses track of the object is counted, and the probability of keeping track of the object at each time slice is calculated based on 1000 simulation runs.

In each time slice, there are 64 motion features available for extraction, one for each grid position. In order to evaluate the ability of our architecture to select efficient features, we restrict the number of features that can be extracted in each time slice. The results of extracting 7, 8, 16, and 64 motion features per time slice, using our pooled PFs architecture enhanced with the particle exchange scheme, are shown in Figure 7.

When 16 motion features are extracted per time slice 25 percent of the features are extracted, with a corresponding reduction in processing resource usage. Still, no significant

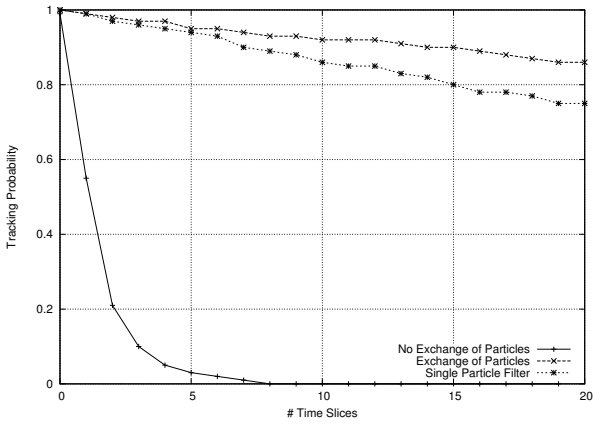


**Figure 8: The tracking probability after # time slices when 8 features are extracted per time slice.**

loss in classification accuracy is observed. This is probably because only motion features which are likely to be in close vicinity of the tracked object are extracted, and 16 motion features are more than sufficient when the object only is allowed to move to adjacent grid positions from time slice to time slice.

When 8 motion features can be extracted per time slice (12.5 percent of the features), the quality of the feature selection technique becomes critical. Generally, there is only one possible configuration of 8 features which allows accurate tracking, namely, the 8 motion features which surround the true (actual) position of the moving object. Still, as seen in the figure, the tracking is quite accurate compared to extracting all 64 features in each time slice.

When only extracting 7 motion features per time slice, the PFs quickly loose track of the object. This is most likely because the object then no longer can be completely enclosed by extracted motion features.

In Figure 8 we compare the results of the pooled PFs architecture with and without the particle exchange scheme. We also test a non-parallel *single* PF that maintains 128 particles for comparison purposes. The results shown in the figure confirm our reasoning from the previous section regarding fragmentation of particles. When a PF in the pool is isolated from the other PFs, the probability that it looses track of the situation is 0.52, at any given time step. However, when our particle exchange scheme is introduced, the globally most probable particle is shared between all the PFs, and as seen, far better accuracy is achieved. Indeed, the pooled PFs architecture, when applying the particle exchange scheme, tracks the object more accurately than the single non-parallel PF. This is despite the fact that the number of particles totals to 128 in both cases. An explanation of this surprising result seems to be found in the field of genetic algorithms, where the population of individuals sometimes are subdivided into so-called demes. A restricted migration process between demes allows more diverse populations to evolve. This is because the *currently* most fit individuals are not allowed to dominate the complete population immediately [19]. A full study of this effect for our pooled PFs architecture is further work.

We have also tested the scalability of our parallel hypothesis driven video content analysis architecture. Firstly, we have implemented an object tracking application in the C programming language, based on the parallel feature extraction and classification phases described in Section 3 [4]. The processing rates achieved when executing respectively 1, 2, 4, and 5 pooled PFs in a switched LAN were measured, and we observed that the processing rate increased linearly with the number of CPUs (see Table 1). Secondly, in a Python implementation of the feature selection and classification phases, the pooling system uses about 7.1 percent of the overall feature selection and classification processing time, as CPUs are added to the simulation. Another object tracking experiment demonstrates that even a non-parallel PF allows effective feature selection and classification in simple settings, at 55 video frames per second on a 933MHz Pentium III CPU [11]. Note that in the latter experiment, feature interactions were also considered, making feature selection computationally more costly.

**Table 1: The achieved frame rate for different numbers of PFs (CPUs) — $20 \times 16$ motion features are extracted per time slice and 1100 particles are shared between the PFs for video frame classification.**

| | 1 PF | 2 PFs | 4 PFs | 5 PFs |
|---|---|---|---|---|
| Frame rate | 5 | 10 | 20 | 25 |

# 5. CONCLUSION

The limited processing resources available on a typical host restrict the complexity, accuracy, and timeliness of video content analysis. In this paper, we have integrated the parallel and hypothesis driven feature extraction and classification approaches. The PF has been modified for real-time hypothesis driven feature extraction in DBNs, and selected features are extracted in parallel to support computationally expensive signal processing. We use a pooling system to coordinate output from multiple PFs. A loss of classification accuracy, caused by parallelization, is avoided by allowing the pooled PFs to exchange particles. Equally important, the processing rates of the techniques seem to increase linearly with the number of CPUs, indicating a feasible solution to the targeted processing bottleneck problem. Finally, by only extracting the most efficient features, the signal processing resource usage can in many cases be reduced, while the classification accuracy is maintained.

# 6. ACKNOWLEDGEMENTS

I would like to thank Viktor Sigurd Wold Eide, Finn Verner Jensen, Frank Eliassen, and Olav Lysne for contributing to ideas presented in this paper.

# 7. REFERENCES

[1] S.-F. Chang and H. Sundaram. Structural and Semantic Analysis of Video. In *Proceedings of Multimedia and Expo (ICME 2000)*, volume 2, pages 687–690. IEEE, 2000.

[2] A. Doucet, V. Ba-Ngu, C. Andrieu, and M. Davy. Particle filtering for multi-target tracking and sensor management. In *Proceedings of the Fifth International Conference on Information Fusion*, volume 1, pages 474–481, July 2002.

[3] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience. John Wiley and Sons, Inc., 2000.

[4] V. S. W. Eide, F. Eliassen, O.-C. Granmo, and O. Lysne. Scalable Independent Multi-level Distribution in Multimedia Content Analysis. In *Protocols and Systems for Interactive Distributed Multimedia (IDMS/PROMS 2002)*, volume 2515 of *LNCS*, pages 37–48. Springer, 2002.

[5] V. S. W. Eide, F. Eliassen, O.-C. Granmo, and O. Lysne. Supporting Timeliness and Accuracy in Distributed Real-time Content-based Video Analysis. In *Proceedings of the 11th Annual ACM International Conference on Multimedia (MM'03)*, pages 21–32. ACM Press, 2003.

[6] V. S. W. Eide, F. Eliassen, O. Lysne, and O.-C. Granmo. Extending Content-based Publish/Subscribe Systems with Multicast Support. Technical Report 2003-03, Simula Research Laboratory, July 2003.

[7] A. M. Ferman and A. M. Tekalp. Probabilistic Analysis and Extraction of Video Content. In *Proceedings of IEEE ICIP*, volume 2, pages 91–95. IEEE, 1999.

[8] A. Francois and G. Medioni. A Modular Software Architecture for Real-Time Video Processing. In *Proceedings of the Second International Workshop on Computer Vision Systems (ICVS 2001)*, volume 2095 of *LNCS*, pages 35–49. Springer, 2001.

[9] A. Garg, V. Pavlovic, and J. M. Rehg. Audio-Visual Speaker Detection Using Dynamic Bayesian Networks. In *Proceedings of the 4th IEEE International Conference on Automatic Face and Gesture Recognition (FGR 2000)*, pages 384–390. IEEE, 2000.

[10] O.-C. Granmo, F. Eliassen, O. Lysne, and V. S. W. Eide. Techniques for Parallel Execution of the Particle Filter. In *Proceedings of the 13th Scandinavian Conference on Image Analysis (SCIA2003)*, volume 2749 of *LNCS*, pages 938–990. Springer, 2003.

[11] O.-C. Granmo and F. V. Jensen. Real-time Hypothesis Driven Feature Extraction on Parallel Processing Architectures. In *Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), Las Vegas, USA*, pages 727–733. CSREA Press, June 2002.

[12] S. Hongeng, F. Bremond, and R. Nevatia. Bayesian Framework for Video Surveillance Applications. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 1, pages 164–170. IEEE, 2000.

[13] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[14] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Series for Statistics for Engineering and Information Science. Springer Verlag, 2001.

[15] F. V. Jensen, H. I. Christensen, and J. Nielsen. Bayesian Methods for Interpretation and Control in Multi-agent Vision Systems. In *Proceedings of Applications of Artificial Intelligence X: Machine Vision and Robotics, SPIE*, 1992.

[16] B. Li and R. Chellappa. A generic approach to simultaneous tracking and verification in video. *IEEE Transactions on Image Processing*, 11:530–544, May 2002.

[17] J. Liu et al. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998.

[18] L. Marcenaro, F. Oberti, G. L. Foresti, and C. S. Regazzoni. Distributed Architectures and Logical-Task Decomposition in Multimedia Surveillance Systems. *Proceedings of the IEEE*, 89(10):1419–1440, October 2001.

[19] T. M. Mitchell. *Machine Learning*. Computer Science Series. McGraw-Hill International Editions, 1997.

[20] A. V. Nefian. Embedded Bayesian networks for face recognition. In *Proceedings of the 2002 IEEE International Conference on Multimedia and Expo*, volume 2, pages 133–136. IEEE, 2002.

[21] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2003.

[22] A. S. Tanenbaum and M. Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2002.