1D and Pseudo-2D Hidden Markov Models for Image Analysis

Implementation details

Stéphane Marchand-Maillet - Multimedia Communications
Email: Stephane.Marchand@Eurecom.fr
Phone: +33 (0)4.93.00.26.79 - Fax: +33 (0)4.93.00.26.27

Date: March 4, 1999
Confidential: No

Eurécom's research is partially supported by its industrial members: Ascom, Cegetel, France Telecom, Hitachi, IBM France, Motorola, Swisscom, Texas Instruments, and Thomson CSF.

EURECOM
Multimedia Communications

Institut EURECOM – BP 193.
06904 Sophia Antipolis – France
T.R. RR-99-49 Part B – March 4, 1999

1D and Pseudo-2D Hidden Markov Models for Image Analysis Implementation details

Stéphane Marchand-Maillet

Stephane.Marchand@Eurecom.fr
http://www.eurecom.fr/~bmgroup/

Institut EURECOM
Department of Multimedia Communications
Technical Report RR-99-49 Part B
March 4, 1999

Abstract

This document constitute the second part of a three-fold report. It follows from [1] where theoretical details about 1D and Pseudo-2D HMM were presented. In this report, we give details about an example package we implemented in view of operating face localisation and detection in 2D images. Based on this application, this documents details main pitfalls generally encountered and proposes solutions to overcome them.

Extending this study, part Three [2] illustrates the capabilities of HMM for performing person-based video segmentation.

Résumé

Ce document forme la deuxième partie d'un rapport qui est contient trois. Il suit [1] où la théorie relative aux Modèles de Markov Cachés (HMM) a été présentée pour le cas de modèles mono- et pseudo-bidimensionnels. Ce rapport fournit les détails d'un exemple d'implémentation d'une telle technique dans le cadre de la détection et de la localisation de visages dans des images bidimensionnelles. Ceci nous permet de traiter avec plus de détail les pièges à éviter pour l'application pratique de ces outils.

La troisième partie [2] de ce rapport se propose d'appliquer ces outils théoriques et logiciels à l'analyse d'images couleur.

Keywords: Image analysis, Video analysis, Stochastic Modelling, Hidden Markov Models

Stéphane Marchand-Maillet. *1D and Pseudo-2D Hidden Markov Models for Image Analysis Implementation details.* Tech. Rep. RR-99-49 Part B, Institut EURECOM, Multimedia Communications, March 4, 1999.

Contents

Introduction

In [1], we introduced the use of Hidden Markov Models (HMM) for 1D and 2D data modelling. The reader is strongly recommended to have a look at this first part for the introduction of the notation we use here.

In this report, the aim is to see how theretical concepts map onto practical implementation. Chapter 1 details the implementation of a Markov Modelling system we put together to experiment our models. As a first step, this code has been written in C. C-structures used are detailed in this report. The object-like structure of HMM is highlighted and we suggest that an efficient implementation may be achieved in C++ or any other object-oriented language.

For the sake of completness and illustration, Chapter 2 gives the results of a practical application of our code onto artificial data.

Chapter 1

Detailed implementation

Based on the specific application of locating faces in video sequences, we have developed a C program that allows for (pseudo) two-dimensional Markov Modelling. Although directed toward image processing application the structure of the program has been kept generic where possible.

1.1 Data representation

Structures have been used for representing both PDF, states and super-states. Due to the similarity between these levels, corresponding structures are similar and embedded one into another. The aim is to be able to remove redundancy in calculations and to allow for flexibility when describing a P2DHMM.

A first structure details the parameters of the input data stream

```
typedef struct            /* Input data stream */
{
 unsigned int globalDim;  /* Total stream dimension */
 unsigned int nStream;    /* Number of sub-streams */
 char **streamLabel;      /* Sub-stream names */
 unsigned int *streamDim; /* Sub-stream dimensions */
 unsigned int *startIndex;/* Sub-stream start indices within the global stream */
}multiStreamInfo_t;
```

This structure will allow for easily carrying this information throughout the datastructure.

As detailed earlier, each stream is associated with its own Gaussian Mixture. The basis for this structure is the multi-dimensional Gaussian structure defined as follows:

```
typedef struct                /* Multi-dimensional Gaussian */
{
 multiStreamInfo_t *streamInfo; /* Info about input stream */
 unsigned int streamIndex;      /* Index of the sub-stream in question */
 FLOAT *meanVect;               /* Mean vector of the Gaussian */
 FLOAT **varMat;                /* Variance matrix of the Gaussian (LU form) */
 FLOAT detVarMat;               /* Determinant of the variance matrix */
}PDF_t;
```

For each sub-stream, a mixture of such Gaussian is to be defined using the following structure:

```
typedef struct                  /* Stream modelisation */
{
 multiStreamInfo_t *streamInfo; /* Info about input stream */
 char *label;                   /* Sub-stream name */
 unsigned int streamIndex;      /* Sub-stream index (in the global stream)*/
 unsigned char *inSubState;     /* Whether this sub-stream acts in a given state */
 unsigned int nMixture;         /* Number of Gaussian */
```

```
 FLOAT *mixCoef;                 /* Mixture coefficients c_jsm*/
 PDF_t *PDF;                     /* Array of Gaussian */
 char init[10];                  /* Initialisation technique considered */
}streamPDF_t;
```

By definition, the output PDF for a given state is a weighted sum of stream PDF. This is translated in the following structure.

```
typedef struct                  /* Output PDF for a state */
{
 multiStreamInfo_t *streamInfo; /* Info about input stream */
 unsigned int outPDFIndex;      /* Index of the output PDF (in the list) */
 char streamStatus;             /* Whether weights are to be updated or otherwise */
 char *label;                   /* Output PDF name (ie state name)
 FLOAT *streamWeight;           /* Array of weights w_js */
 streamPDF_t **streamPDF;       /* Array of pointer on stream PDF */
 char init[10];                 /* Initialisation technique considered */
}outPDF_t;
```

A structure is used to represent a generic super-state λ^i (*i.e.*, a 1DHMM). This structure is given as follows:

```
typedef struct                  /* 1D-HMM modelling a superstate */
{
 multiStreamInfo_t *streamInfo; /* Info about input stream */
 unsigned int nState;           /* Number of states */
 unsigned char *label;          /* Name of the superstate */
 FLOAT *initStateProb;          /* Initial state probabilities */
 FLOAT **transProb;             /* Transition probability matrix */
 outPDF_t **outPDF;             /* Array of output PDF corresponding to states */
 char init[10];                 /* Initialisation technique considered */
}HMM_t;
```

The fact of storing respective output PDF associated to each state within a super-state as an array of pointers (*i.e.*, `outPDF_t **outPDF;`) allows for associating the same output PDF to different states. This is useful when it is known that two different states actually model the same type of observation (*e.g.*, the background, see example next). This ensures that output PDF are exactly equal and avoid redundancy, both in storage and computation.

This technique is used again in the structure that represent the upper-level structure of the P2DHMM.

```
typedef struct                  /* P2D-HMM for modelling a 2D structure */
{
 multiStreamInfo_t streamInfo;  /* Info about input stream */
 unsigned int nState;           /* Number of superstates */
 char *label;                   /* Name of the model */
 unsigned int maxSubState;      /* Maximum number of states in a superstate */
 unsigned int maxMixture;       /* Maximum number of mixtures in a stream PDF */
 FLOAT *initStateProb;          /* Initial state probabilities */
 FLOAT **transProb;             /* Transition probability matrix */
 HMM_t **hmm;                   /* Array of 1D-HMM pointers corresponding to superstates */
 unsigned int nCommonHMM;       /* Number of 1D HMM defined in the complete model */
 HMM_t *commonHMM;              /* Array storing the 1D HMM defined in the model */
 unsigned int nCommonOutPDF;    /* Number of output PDF defined in the model */
 outPDF_t *commonOutPDF;        /* Array storing the output PDF defined in the model */
 unsigned int nCommonStreamPDF; /* Number of stream PDF defined in the model */
 streamPDF_t *commonStreamPDF;  /* Array storing the stream PDF defined in the model */
 char init[10];                 /* Initialisation technique considered */
}PHMM_t;
```

The array `HMM_t **hmm` stores pointers on HMM structures so that different super-states are kept exactly equivalent during the processing. All possible PDF and HMM are stored in arrays `PDF_t *commonOutPDF`, `PDF_t *commonStreamPDF` and `HMM_t *commonHMM` respectively. Elements of the array `streamPDF` in an output PDF (*i.e.*, `outPDF_t`) structure will be taken from the array `commonStreamPDF`. Similarly, elements of the array `outPDF` in a HMM (*i.e.*, `HMM_t`) structure will be taken from the array `commonOutPDF` and elements of the array `hmm` in a P2DHMM (*i.e.*, `PHMM_t`) structure will be taken from the array `commonHMM`. Re-estimation therefore focuses on updating values in arrays `commonStreamPDF`, `commonOutPDF` and `commonHMM` so that computation is reduced to minimum. Parameters `maxSubState` and `maxMixture` represent the maximal number of state in a super-state (*i.e.*, $\max_i N^i$) and the maximal number of mixture in a stream PDF (*i.e.*, $\max_{i,j} M_j^i$) and are stored explicitly for easy accessibility.

Example 1 P2DHMM for face localisation.
An example of P2DHMM structure that can be represented using these data-structures is shown in Figure 1.1.

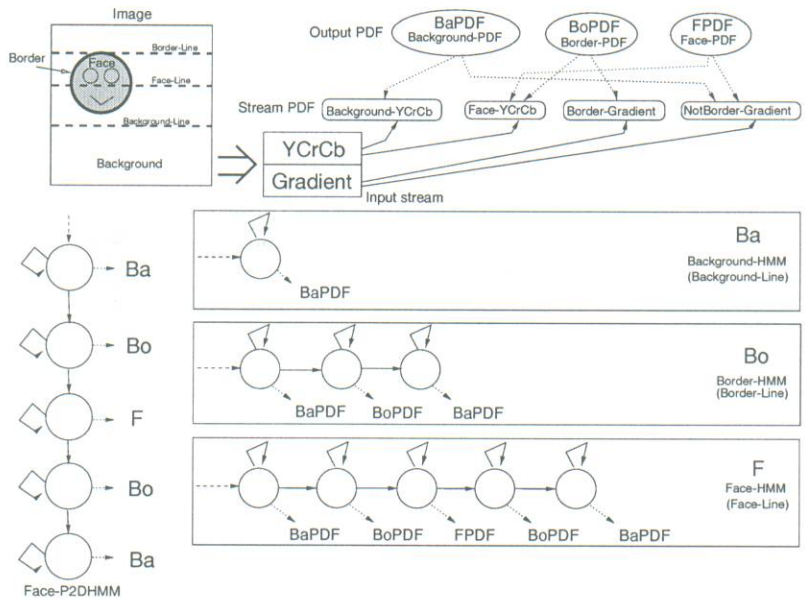

Figure 1.1: A Pseudo-2D Hidden Markov Model used for face location.

In this application each line of a face image will be model by a super-state in our P2DHMM. Three types of pixels are distinguished: Face, Border, and Background. Three output PDF Face-PDF Border-PDF and Background-PDF will therefore represent two states (*i.e.*, being a Face, Border or a Background pixel, respectively). Similarly, three types of sequences (*i.e.*, lines) are distinguished: Face-Line (*i.e.*, a line containing a part of a face), Border-Line (*i.e.*, a line containing background and border pixels) and Background-Line (*i.e.*, a line containing background pixels only). Hence, three HMM structures, Face-HMM, Border-HMM, and Background-HMM will represent these three possibilities. These elements will therefore form arrays `commonPDF` and `commonHMM` respectively. Now, the input stream is composed of two parts (assumed to be independent). Three components form the YCrCb vector and one component forms the Gradient vector. The output PDF Face-PDF will be a combination of stream PDF Face-YCrCb and NotBorder-Gradient. The output PDF Border-PDF will be a combination of stream PDF Face-YCrCb and Border-Gradient. The output PDF Background-PDF will be a combination of stream PDF Background-YCrCb and NotBorder-Gradient. This structure is illustrated in Figure 1.1. The structure of the P2DHMM composed of horizontal and vertical Bakis-like models is shown in Figure 1.1 where plain arrows represent state and super-state transitions, dashed arrows represent initial state probabilities and dotted arrows represent (C) pointers. This example clearly illustrates the advantage of keeping some parts of the P2DHMM exactly equivalent. ◇

1.2 Program modules

The part of our program concerning HMM has been kept separated from the rest of the processing so that application to another topic will be facilitated. Typically, this part is essentially composed of the following files:

- `hmm_s.h`: Header file for HMM structures.
- `baum.c`: Baum-Welch related procedures (training, re-estimation).
- `viterbi.c`: L.B.G. algorithm, Viterbi procedure (training, re-estimation).
- `parse.c`,`parse.ycc`,`parse.lex`: reading, parsing and writing of HMM files.
- `hmm.c`: Miscellaneous functions related to HMM calculations (*e.g.*, $\mathcal{N}(o, \mu, \Sigma)$).

1.3 P2DHMM files

A P2DHMM will be described by two separate files, namely a configuration file (extension `.cfg`) and a initialisation file (extension `.ini`). The configuration file stores details on the structure of the P2DHMM whereas the initialisation file contains actual values of P2HMM fields. This allows for using the same structure with different initialisations (*e.g.*, when different feature vectors are considered). For examples of such files, see Appendix A.

1.4 Training a P2DHMM for recognition

The idea of the P2DHMM being to segment 2D data by associating a state to each observation, training is initiated by giving to the P2DHMM a set of data segmented by hand. The set of training data (training set) should be large enough and representative of the data on which recognition will be performed (test set). In the case of face localisation for example, the training set will consist in a set of images where the faces have been explicitly labelled.

Two files are created that store the training data set. `observFile` is the observation sequence stored sequentially with 1 byte per component of observation ($o_{xy}^{(k,d)}$). In other words, the file `observFile` contains the sequence:

$$o_{11}^{(1,1)}, o_{11}^{(1,2)}, \ldots, o_{11}^{(1,D)}, \ldots, o_{XY}^{(1,D)}, \ldots, o_{XY}^{(K,D)}.$$

Each D-dimensional observation $o_{xy}^{(k)}$ is paired with a state label (*i.e.*, the index of the corresponding common PDF modelling this observation). State labels are stored in the state file `stateFile` sequentially using 1 byte per observation. Corresponding state sequence is therefore

$$q_{11}^1, \ldots, q_{XY}^1, q_{11}^2, \ldots, q_{XY}^K$$

In summary, the file `observFile` is of size $XYKD$ bytes and the state file is of size XYK.

A third file `trainFile` stores informations about training data (*e.g.*, number and size of training sequences). This allows for completely dissociating the HMM part to the rest of the interface in case these procedures are to be used in another type of application or possibly in a batch process.

For initialisation, LBG algorithm is used. It allows for finding parameters of the Gaussian mixtures modelling best the given observation while matching characteristics given in the configuration file (*e.g.*, number of mixtures).

Chapter 2

Validation through segmentation of artificial data

2.1 Approach

The first step in validating the use of P2DHMM for segmenting images is to test this approach on artificially generated images. Our approach is as follows.

Instances of a given image model are created via the given of means and variance of observations. The set of such instances is divided into a training set and a test set. Training is operated and tests are reported.

Different features of the approach are to be evaluated:

- Capability of the P2DHMM to retrieve fuzzy boundaries within the image. This will be done independently to multi-stream approach (see next item).
- The multi-stream functionality needs further tests to validate automatic setting of stream weights. Dimensionality is a problem here as detailed below.
- Once the general approach is known to be valid, there is a need for evaluating best components that will form the observation vectors in order to achieve best possible performance in the face location application.

2.2 Segmentation of 2D images

Segmentation is tested using artificial colour face-like images. An instance of such image is shown in Figure 2.1.

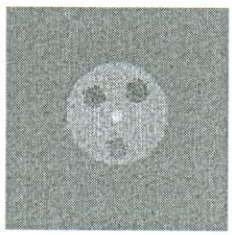

Figure 2.1: Artificial face image.

Means and variances of the RGB tuple corresponding to each part (*e.g.*, eyes, face, nose) are modulated so that limitation of the model are evaluated. A noise component which creates fuzzy boundaries is also added in the images. The model used for segmenting such images is depicted in Figure 2.2.

The figure indicated in superstates link to horizontal models corresponding to different types of lines. Results of the segmentation using images where parts shown clear differences in means

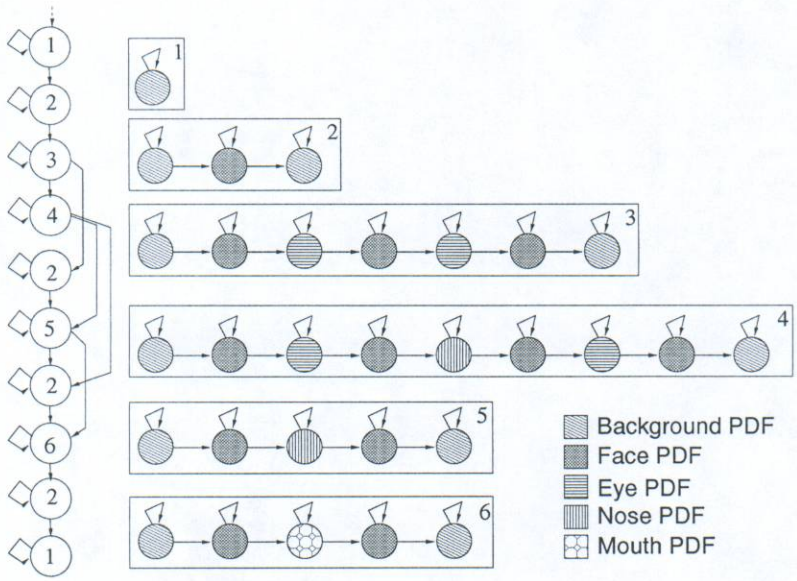

Figure 2.2: Model for a face image.

of RGB tuple is depicted in Figure 2.3. From left to right, images show parts corresponding to Background, Background, Face, Eye, Nose, and Mouth respectively. Clearly these results show the capability of the P2DHMM is segmenting such an artificial image. Component of observations are YCrCb components considered as a 3D input stream.

Figure 2.3: Decomposition of a face image.

Multi-stream functionality is tested on the following instance. An image composed of two main zones is to be segmented. Two states Foreground and Background are therefore distinguished. In order to emphasise the notion of border, a third Border state is created. The input stream is composed of two parallel stream. One with the luminance (Y) component and one with the gradient norm ($||\nabla I||$). The model depicted in Figure 2.4 is used for the segmentation.

Before training stream weights are equal and balanced ($w_{js}^{i} = 0.5$) for all states. After training weights adapt the influence of each stream for a given state. Results are typically as follows (note that during the initialisation the Border state follows the same stream first component (Y) as the Foreground state):

$$w_{\mathrm{Y}}^{\mathsf{Foreground}} = 0.2 \; ; \; w_{||\nabla I||}^{\mathsf{Foreground}} = 0.8$$

$$w_{\mathrm{Y}}^{\mathsf{Background}} = 0.67 \; ; \; w_{||\nabla I||}^{\mathsf{Background}} = 0.33$$

$$w_{\mathrm{Y}}^{\mathsf{Border}} = 0.05 \; ; \; w_{||\nabla I||}^{\mathsf{Border}} = 0.95$$

Resulting decomposition is shown in Figure 2.5. A blurring with a Gaussian kernel ($\sigma = 4$ pixels) is operated. This example shows the capability of multi stream functionality to adapt to the context. However, a problem arises when stream do not have the same dimensionality. High weights seem to be set for streams of lower dimensions. For example, with the same example as above, replacing stream Y by the 3D stream YCrCb in parallel with $||\nabla I||$ we obtain

$$w_{\mathrm{YCrCb}}^{\mathsf{Foreground}} = 0.01 \; ; \; w_{||\nabla I||}^{\mathsf{Foreground}} = 0.99$$

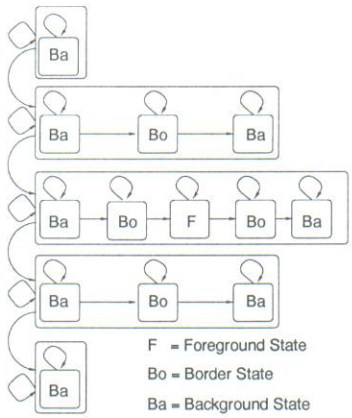

Figure 2.4: Model a simple image.

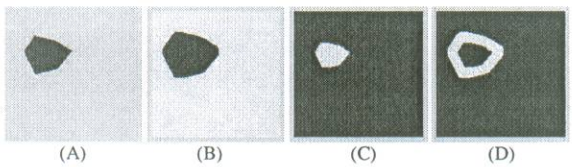

Figure 2.5: Decomposition of an image. (A) Original. (B) Background. (C). Foreground. (D) Border.

$$w_{\mathrm{YCrCb}}^{\mathsf{Background}} = 0.01 \; ; \; w_{\|\nabla I\|}^{\mathsf{Background}} = 0.99$$

$$w_{\mathrm{YCrCb}}^{\mathsf{Border}} = 0.01 \; ; \; w_{\|\nabla I\|}^{\mathsf{Border}} = 0.99$$

which does not correspond to any reality. This is emphasised by the fact that the segmentation obtained in wrong. Multi-streaming therefore seems correct under the condition that streams have the same dimension.

Conclusion on the pratical developments

In this part of our study of HMM, we presented an approach for implementing HMM and (P2DHMM). This analysis, albeit in C, should provide the reader with enough material for approaching the problem in an object-oriented manner (*e.g.*, using C++). We are confident that our structures can easily be mapped onto objects.

In summary, the aim here is two-fold. Firstly, this report details the existing code and should give the reader insights as to how to handle HMM programming.

Appendix A

P2DHMMfiles

A.1 Configuration file

This file is used for describing the structure of the P2DHMM. Objects corresponding to C structures described in Chapter 1 are successively detailed with a consistent syntax.

```
# bistream.cfg
# Simple Model for face localisation
# Using YCC and Gradient stream
# separately
#
# (Anything after a # is a comment)
#
StreamDescription:
{
 Size: 3 1      # Two part stream
 Label: YCC G
}
StreamPDF:
{
 Component: YCC
 Label: YCC_Background
 NMixture: 3
}
StreamPDF:
{
 Component: YCC
 Label: YCC_Face
 NMixture: 3
}
StreamPDF:
{
 Component: G
 Label: G_Edge
 NMixture: 1
}
StreamPDF:
{
 Component: G
 Label: G_NonEdge
 NMixture: 3
}

OutputPDF:
{
 Label: Background
 Streams: YCC_Background G_NonEdge
 Status: FREE     # or LOCKED to a value
}
OutputPDF:
{
 Label: Face
 Streams: YCC_Face G_NonEdge
 Status: FREE     # or LOCKED to a value
}
OutputPDF:
{
 Label: Border
 Streams:  YCC_Face Edge
 Status: FREE     # or LOCKED to a value
}
HMM:
{
 Label: Face_Line
 NState: 5
 State:
 {
  Index: 1
  PDF: Background
  Transition: 1 2
 }
 State:
 {
  Index: 2
  PDF: Border
  Transition: 2 3
 }
 State:
 {
  Index: 3
  PDF: Face
  Transition: 3 4
 }
 State:
 {
  Index: 4
  PDF: Border
  Transition: 4 5
 }
 State:
 {
  Index: 5
  PDF: Background
  Transition: 5
 }
}
HMM:
{
 Label: Border_Line
 NState: 3
 State:
 {
  Index: 1
  PDF: Background
  Transition: 1 2
 }
 State:
 {
```

```
  Index: 2
  PDF: Border
  Transition: 2 3
 }
 State:
 {
  Index: 3
  PDF: Background
  Transition: 3
 }
}
HMM:
{
 Label: Background_Line
 NState: 1
 State:
 {
  Index: 1
  PDF: Background
  Transition: 1
 }
}
P2DHMM:
{
 Label: Simple_Grad_Model
 NSuperState: 5
 SuperState:
 {
  Index: 1
  HMM: Background_Line
  Transition: 1 2
 }
 SuperState:
 {
  Index: 2
  HMM: Border_Line
  Transition: 2 3
 }
 SuperState:
 {
  Index: 3
  HMM: Face_Line
  Transition: 3 4
 }
 SuperState:
 {
  Index: 4
  HMM: Border_Line
  Transition: 4 5
 }
 SuperState:
 {
  Index: 5
  HMM: Background_Line
  Transition: 5
 }
 InitProb: 1
}
include: <bistream.ini>
```

A.2 Initialisation file

This file is used to associate each object with an initialisation technique. Once the structure is created reading the `.cfg` file, fields are filled with values using initialisation technique specified here.

```
# bistream.ini
# Simple Model for face localisation
# Using YCC and Gradient stream
# separately
StreamPDF: YCC_Face -> TRAINING
StreamPDF: YCC_Background -> RANDOM
StreamPDF: G_Edge -> VALUES
{
 Mixture: 1
 {
  Coef: 1.
  Mean: 250.
  VarMat:
        25.
 }
}
StreamPDF: G_NonEdge -> TRAINING
OutputPDF: Face -> VALUES # 2 Streams (FREE)
{
 Weights: 0.933102 0.066898
}
OutputPDF: Background -> TRAINING
OutputPDF: Border -> EQUAL # equal weights

HMM: Background_Line -> STANDARD 338
      # 1 state change out of n
HMM: Face_Line -> TRAINING
HMM: Border_Line -> VALUES
{
 Transition:
  0.991921 0.008079 0.000000
  0.000000 0.990299 0.009701
  0.000000 0.000000 1.000000
}
P2DHMM: Simple_Grad_Model -> STANDARD 274
      # or RANDOM or TRAINING or VALUES
}
```

Bibliography

[1] S. Marchand-Maillet. 1D and pseudo-2D Hidden Markov Models for image analysis – A: Theoretical introduction. Technical report, EURECOM Institute – Dept of Multimedia Communications, 1999.

[2] S. Marchand-Maillet. 1D and pseudo-2D Hidden Markov Models for image analysis – C: Applications and results. Technical report, EURECOM Institute – Dept of Multimedia Communications, 1999.