Exercises for
Pattern Analysis - Programming Assignments
Dalia Rodriguez Salas & Daniel Stromer
Assignment #4

General Information:

Lecture (3 SWS):	Thu 12.15 – 13.45 (H16) and Tue 12.15 – 13.45 (H16)
Exercises (1 SWS):	Tue 14.00 – 16.00 (02.151b-113) and Wed 10.00 – 12.00 (02.151b-113s)
Certificate:	Oral exam at the end of the semester
Contact:	daniel.stromer@fau.de & dalia.rodriguez@fau.de

Hidden Markov Models

Exercise 1 Rules for submitting the programming exercise:

(a) Work together in pairs (max. two people).

(b) You have to show your code to the tutors not later than the deadline. It's is recommended that all team members shows up.

(c) Your code has to be in C or C++. We recommend using OpenCV for Matrix algebra and visualization, as it is available in the CIP pool.

(d) You can either use CIP pool PC's or your own laptop.

(e) Plagiarism will be punished by assigning zero points, removal from the programming exercises and/or by a report to the examination office. According to Wikipedia, plagiarism is *the "wrongful appropriation" and "stealing and publication" of another author's "language, thoughts, ideas, or expressions" and the representation of them as one's own original work.*

Exercise 2 Programming exercise:

The goal of this exercise is to implement two algorithms relating to Hidden Markov Models (HMMs).

(a) Download *main_students.cpp* from Studon. This file contains parameters of a HMM, as well as output code for results. The states and symbols are encoded using integers. This model has four states and three output symbols.

(b) HMMs are generative models. As you have learned in the lecture, its is possible to sample from a generative model. We already implemented the *generateRandomObservations* function for you. This function generates a random sequence of *observationCount observations* from the given HMM. We used the predefined randomization function for obtaining uniformly distributed values between zero and one. Check this function to get a feeling for the following implementations.

(c) Implement *observationProbabilityForward*. This function should use the Forward algorithm to compute the probability that a given HMM produced the

observation sequence in *observations*.

(d) Implement the *bestStateSequence*. This function should use the Viterbi algorithm to compute the most likely state sequence of the HMM produced the given observations. The sequence should be stored in *bestStates*. The function should return the probability of observing the observations when taking this state sequence.

(e) **Deadline for submission: 10th and 11th of July**