TP : Chaînes de Markov cachées

Objectifs

— se familiariser avec les chaînes de Markov cachées (Hidden Markov Chains - HMM).
— mettre en application les chaînes de Markov cachées pour la reconnaissance de gestes

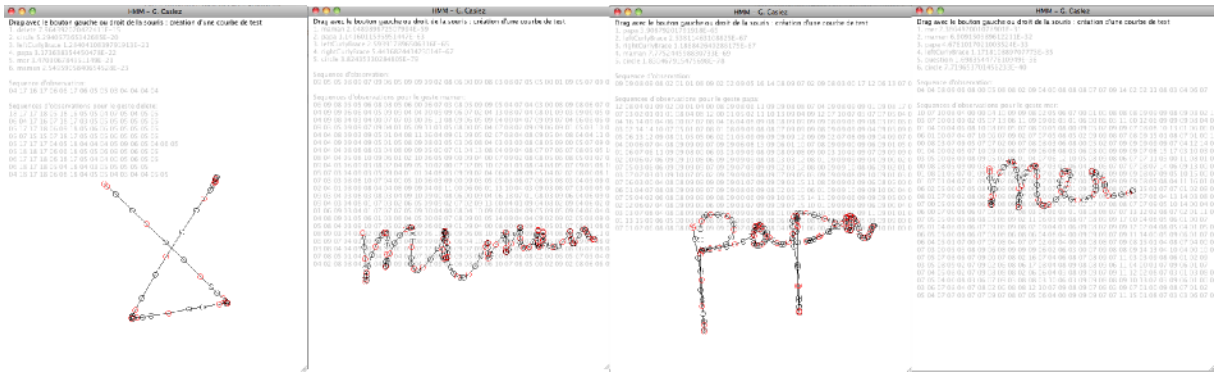

FIGURE 1 – Différentes captures d'écrans de l'application finale.

1 Matériel à disposition

Une interface écrite en Java Swing est mise à votre disposition pour saisir des courbes en utilisant la souris. Cette interface comprend les fichiers TestGUI.java [1] pour définir la fenêtre de l'application et Canvas2D.java [2], qui hérite de Canvas [3], pour définir la zone de dessin. L'interface vous permet de saisir et afficher (en noir) une courbe de test en cliquant sur un des boutons de la souris. La courbe tracée sera ensuite reconnue en utilisant les chaînes de Markov cachées.

Un ensemble de 10 exemples de 16 gestes de référence (Figure 2) est également mis à votre disposition dans le fichier gesture.xml [4]. Ces gestes sont les mêmes que ceux utilisés dans l'article "Gestures without Librairies, Toolkits or Training: A $1 Recognizer for User Interface Prototypes" [5] écrit par Wobbrock, Wilson et Li et publié en 2007 à la conférence User Interface Sofware and Technology (UIST). Chaque exemple de geste est appelé *template*. Un geste de référence est appelé une classe de geste (appelé *GestureClass* dans la suite). Nous avons donc 10 *templates* par *GestureClass*. La classe TemplateManager [6] permet de charger tous les *templates* du fichier xml. Les *templates* sont représentés par la classe Template [7]. Un *template* enregistre l'ensemble des coordonnées (x,y) du geste tracé. Une étiquette de temps (*timestamp*) est également associée à chaque point. Ces trois informations sont représentées par la classe PointData [8]. Une classe de geste est représentée par la classe GestureClass [9]. Vous aurez besoin de la librairie JDOM (fichier jdom.jar [10] dans l'archive zip). A ces 16 gestes issus de $1 s'ajoutent 3 gestes correspondants à de l'écriture manuscrite : "maman", "papa", et "mer" (Figure 1).

1. TestGUI.java
2. Canvas2D.java
3. http://docs.oracle.com/javase/7/docs/api/java/awt/Canvas.html
4. gestures.xml
5. http://depts.washington.edu/aimgroup/proj/dollar/
6. TemplateManager.java
7. Template.java
8. PointData.java
9. GestureClass.java
10. Dans Eclipse, l'ajout de librairies au projet se fait en allant dans Build Path > Add External Archives... et en sélectionnant les fichiers jar à ajouter

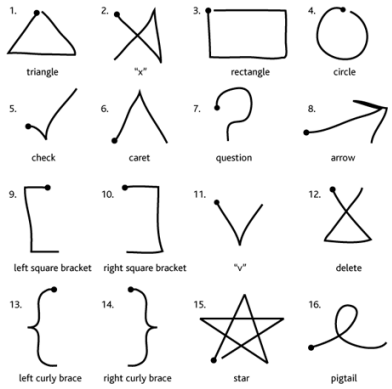

FIGURE 2 – Les 16 gestes de référence utilisés pour la reconnaissance.

2 La librairie jahmm

Nous utilisons la librairie jahmm [11] comme implémentation des chaînes de Markov cachées en Java (fichier jahmm-0.6.1.jar dans l'archive zip). La librairie implémente par ailleurs les algorithmes suivant de l'état de l'art : K-Means, Baum-Welch, Forward-Backward et Viterbi.

- Rabiner, Juang, An introduction to Hidden Markov Models, IEEE ASSP Mag., pp 4-16, June 1986
- Juang, Rabiner, The segmental k-means algorithm for estimating the parameters of hidden Markov Models, IEEE Trans. ASSP, vol. 38, no. 9, pp. 1639-1641, Sept. 1990.
- Juang, Rabiner, A Probabilistic distance measure for HMMs, AT&T Technical Journal, vol. 64, no. 2, pp. 391-408, Feb. 1985.
- Juang, Rabiner, Fundamentals of speech recognition, Prentice All, AT&T, 1993.

La documentation de la librairie jahmm est diponible ici [12].
La documentation des différentes classes écrites pour le TP est disponible ici [13].

3 Procédure de d'apprentissage et de reconnaissance

D'une façon générale, l'apprentissage et la reconnaissance d'un phénomène suivent les étapes illustrées sur la figure 3. Premièrement des données représentant le phénomène observé sont acquises (observations) avant d'être pré-taitées pour être mises en forme. Des caractéristiques (*features*) sont ensuite extraites et comparées aux caractéristiques de référence par le système de reconnaissance (*pattern recognition*). Une étape de post-traitement permet de réduire les faux positifs.

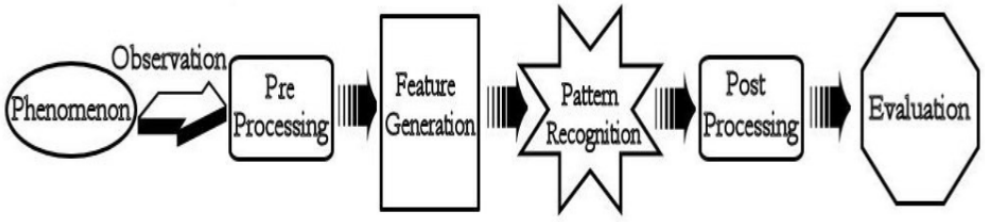

FIGURE 3 – Procédure d'apprentissage et de reconnaissance.

La classe HMM [14] réalise ces différentes étapes.

11. http://code.google.com/p/jahmm/
12. jahmm-0.6.1-javadoc/index.html
13. doc/index.html
14. HMM.java

4 Pré-traitement

Le pré-traitement consiste à ré-échantilloner les points pour qu'ils soient espacés à intervalle de temps régulier.

Question 1. Remplissez la méthode `resample` de la classe `HMM` pour effectuer ce travail. Les points que vous créerez seront des interpolations linéaires en x et y des points sources. Vous pourrez observer le résultat qui est représenté par les points rouges à l'écran.

5 Extraction de caractéristiques

Pour chacun des points pré-traités, nous allons extraire une caractéristique. Pour faire simple, nous choisissons de calculer l'angle formé entre le point précédent, le point courant et l'horizontal. Nous prendrons la valeur absolue de cet angle, arrondi à 10 degrés près. On obtient donc un entier entre 0 et 18. C'est notre observation du phénomène. L'ensemble des observations pour tous les points du geste constitue une séquence d'observation.

Question 2. Remplissez la méthode `computeFeatures` pour réaliser cette opération. La méthode `Math.atan2` vous sera utile pour calculer les angles. Les features seront stockées dans une hashmap, dont la clé est la chaine `"fx"` avec `x` l'indice de la feature.

Question 3. Quels sont les défauts de la méthode d'extraction proposée ?

6 Apprentissage

L'algorithme KmeansLearner est utilisé pour la phase d'apprentissage. Une chaîne de Markov est associée à chaque classe de geste (voir GestureClass.java [15])

Question 4. Lisez et comprenez les méthodes `computeKmeansLearner` et `trainHMM` de `GestureClass`.

7 Reconnaissance

La reconnaissance consiste à effectuer, pour le geste candidat, les mêmes pré-traitements et extraction de caractéristiques que pour les templates des classes de gestes. La séquence d'observations obtenue par l'extraction de caractéristiques est alors comparée aux chaînes de Markov cachées des différentes classes de gestes. La comparaison consiste à estimer la probabilité que la séquence d'observation soit le résultat de la chaîne de Markov, en utilisant l'algorithme Forward-Backward.

Question 5. Etudiez la documentation de jahmm pour voir comment utiliser l'algorithme Forward-Backward. Complétez la méthode `computeScore` de `GestureClass`.

Question 6. Testez le tout ! (Dé-commentez la ligne `Training` à la fin du constructeur de `HMM` et `hmm.recognize()` dans `Canvas2D`).

8 Post-Traitement

La post-traiement permet de détecter les faux positifs : par exemple un geste tracé qui n'a rien à voir avec les différentes classes de gestes. Une première solution consiste à mettre un seuil sur la probabilité calculée. Une seconde solution consiste à calculer des caractéristiques globales sur le geste (distance entre premier point et dernier point par exemple) et ensuite déterminer si elles sont valides.

Question 7. Testez la première solution.

15. `GestureClass.java`

9 Réglages

Question 8. Jouez avec le nombre d'état cachés des chaînes de Markov, le pas de temps pour le ré-échantillionnage et le calcul des caractéristiques pour observer les résultats sur les performances de reconnaissance. Quelle est l'influence de ces différents paramètres, comment les optimiser ? La méthode `TestAllExamples` de `HMM` peut aider à répondre à cette question.

10 Nouveaux gestes

Question 9. Ajoutez de nouveaux gestes au fichier gestes.xml, en utilisant la méthode `writeRawPoints2XMLFile` de `HMM`. Testez-les !