Incremental Learning for Motion Prediction of Pedestrians and Vehicles

Dizan Alejandro Vasquez Govea

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

THÈSE

pour obtenir le grade de **DOCTEUR DE L'INPG**

Spécialité : « Imagerie, Vision, Robotique »

préparée au laboratoire Gravir dans le cadre de **l'École Doctorale « Mathématiques, Sciences et Technologies de l'Information, Informatique »**

préparée et soutenue publiquement par

Alejandro Dizan Vasquez Govea

le 13 Février 2007

Titre :

Incremental Learning for Motion Prediction of Pedestrians and Vehicles

---

sous la direction de Christian Laugier et Thierry Fraichard

JURY

M. Augustin Lux	Président
M. Roland Siegwart	Rapporteur
M. Michel Devy	Rapporteur
M. Wolfram Burgard	Examinateur
M. Hiromichi Yanagihara	Examinateur
M. Olivier Aycard	Examinateur
M. Christian Laugier	Directeur de thèse
M. Thierry Fraichard	Co-directeur de thèse

To Lota, my son, for being unpredictable

Abstract

The main subject of this thesis is motion prediction. The problem is approached from the hypothesis that the dynamic and kinematic properties of objects such as pedestrian and vehicles do not suffice to predict their motion in the long term. Instead, the work presented here, inscribes itself in a new family of approaches which assume that, in a given environment, objects do not move at random, but engage in "typical motion patterns", which may be learned and then used to predict motion on the basis of sensor data. In this context, this thesis focuses in three fundamental questions: modeling, learning and prediction.

Modeling. This thesis is based on Hidden Markov Models, a probabilistic framework, which is used as a discrete approximation to represent the continuous state-space in which motion takes place. The main originality of the approach lies in modeling explicitly the intentions which are at the origin of "typical motion patterns". This is achieved through the using of an extended space, which adds the state that the object intends to reach to the other "classic" state variables, such as position or velocity.

Learning. The main problem of existing approaches lies in the separation of model learning and utilization in two distinct stages: in a first phase, the model is learned from data; then, it is used to predict. This principle is difficult to apply to real situations, because it requires at least one example of every possible typical pattern to be available during the learning phase.

To address this problem, this thesis proposes a novel extension to Hidden Markov Models which allows simultaneous learning and utilization of the model. This extension incrementally builds a topological map – representing the model's structure – and reestimates the model's parameters. The approach is intended to be general, and it could be used in other application domains such as gesture recognition or automatic landmark extraction.

Prediction. In this context, prediction is carried on by using exact Bayesian inference algorithms, which are able to work in real time thanks to the properties of the structure which has been learned. In particular, the time complexity of inference is reduced from $O(N^2)$ to $O(N)$ with respect to the number of discrete states in the system.

All of the results obtained on this thesis have been implemented and validated with experiments, using both real and simulated data. Real data has been obtained on two different visual tracking systems: one installed over a parking lot, and the other installed at INRIA's entry hall. For synthetic data, a simulator has been developed in order to facilitate the conduction of controlled tests and the study of larger environments than for real data.

Acknowledgements

First of all, I would like to thank Christian Laugier and Thierry Fraichard, my thesis directors, for their guidance, their support, and for bearing with my – sometimes – idiosyncratic work methods.

I would like to thank the members of my jury; in particular Roland Siegwart and Michel Devy, my reviewers, who took the time and energy to point out my errors and omissions. My thanks also go to my examiners Wolfram Burgard, Hiromichi Yanagihara and Olivier Aycard for the interest they manifested on my work. I want to thank Augustin Lux for honoring me by presiding my jury.

I'd also like to thank all of my colleagues at team eMotion with whom I had the pleasure of working over the years. Special thanks go to Thiago Bellardi and Agostino Martinelli for carefully reading my manuscript and for the many corrections and suggestions they made.

I am very grateful to Hannah Dee from the University of Leeds for kindly allowing me to use her data.

I would like to thank Fernando Ramos, whose help and advice are at the very origin of this adventure.

And last, but not least, I want to express all my gratitude to Barbara, my partner, for her support, friendship, and for her enormous patience during all this period, thank you Babalucalas!

Contents

List of Figures

List of Algorithms

Extended Abstract in French

Chapitre 1 : Introduction

La planification des mouvements pour des environnements dynamiques est un domaine de recherche très actif. En raison du fait que le problème est NP-complet, la plupart des efforts de recherche se sont concentrés sur le développement des mécanismes pour gérer cette complexité. Néanmoins, il y a un autre aspect critique du problème qui souvent a été négligé : les algorithmes de planification ont besoin de connaître d'avance les mouvements des objets qui peuplent l'environnement. Étant donné que, dans la pratique, cette information n'est que très rarement disponible, cela implique la nécessité de recourir à l'utilisation de techniques de prédiction pour obtenir, à partir des informations recueillies en utilisant des capteurs tels que des radars et des systèmes de suivi, une estimation de l'état futur des objets.

Jusqu'à récemment, la plupart des techniques de prédiction que l'on trouve dans la littérature se sont basées sur des modèles cinématiques ou dynamiques qui décrivent l'évolution de l'état des objets par rapport au temps quand ils sont soumis à un contrôle donné (ex. accélération). Ces approches procèdent, d'abord, pour estimer l'état actuel de l'objet à l'aide des techniques telles que le filtre de Kalman, après, elles appliquent l'estimation ainsi obtenue aux équations de mouvement pour obtenir des prédictions.

Même si ces techniques sont capables de produire de très bonnes prédictions à court terme, leur performance se dégrade rapidement quand elles essayent de "voir" plus loin dans le futur. Cela est spécialement vrai dans le cas des humains, véhicules, robots, animaux et autres objets, qui sont capables de modifier leurs trajectoires en fonction de facteurs tels que leurs perceptions, leur état interne, et leurs intentions, qui ne sont pas décris par leurs propriétés cinématiques ou dynamiques.

Cette situation a motivé l'émergence, dans la dernière décennie, d'une nouvelle famille d'approches basées dans l'idée que, dans un environnement donné, les objets ont tendance à suivre des mouvements "typiques" qui dépendent de la nature de l'environnement, ainsi que de la nature des objets eux-mêmes. Ces approches opèrent en deux étapes distinctes :

1. Apprentissage : observer les objets dans l'environnement pour identifier et construire des modèles des mouvements typiques.

2. Prédiction : utiliser les modèles appris pour prédire l'état futur d'un objet donné.

D'un autre côté, toutes ces approches partagent aussi un inconvénient : ils opèrent dans un schéma séquentiel que nous appelons "apprendre puis prédire" dans lequel l'apprentissage

s'effectue avant l'utilisation et il est réalisé qu'une seule fois. Cela implique de façon implicite l'hypothèse que, dans l'ensemble des données utilisées pour l'apprentissage, il existe au moins un exemple de chaque comportement à apprendre. Dans la pratique, cette hypothèse est très difficile à vérifier à cause de la grande diversité des comportements qu'il est possible d'observer même dans les environnements les plus simples. Dans cette thèse, nous proposons l'utilisation alternative d'une approche de type "apprendre et prédire" où l'apprentissage et la prédiction sont réalisés de façon parallèle.

1.2 Description du problème

Une approche de prédiction des mouvements basée dans l'apprentissage est constituée, au moins, de trois composantes:

1. Un modèle de mouvement décrivant comme l'état de l'objet évolue à mesure que le temps passe, sachant que l'objet est en train d'exécuter un mouvement typique donné.
2. Un algorithme d'apprentissage, qui spécifie comment les paramètres du modèle sont calculés à partir des données.
3. Un algorithme de prédiction détaillant l'utilisation des modèles appris pour prédire les mouvements futurs.

1.2.1 Modélisation des mouvements en utilisant des Modèles de Markov Cachés

Nous avons choisi l'utilisation des "Modèles de Markov Cachés" (MMC) pour modéliser les mouvements des objets. Les MMC son un outil probabiliste qui permet de représenter de façon explicite l'incertitude qui est inhérente à tout processus de prédiction. Un MMC peut être vu comme un graphe décrivant un processus où les noeuds correspondent à des états discrets et les arêtes représentent la probabilité que le processus passe d'un état à l'autre. Une particularité des MMC est que les transitions ont lieu de façon aléatoire, en suivant une distribution de probabilité donnée.

1.2.2 Défis

Pour pouvoir implémenter de façon efficace une approche "apprendre et prédire" en utilisant des MMC, il est nécessaire de résoudre les points suivants :

1. Apprendre les paramètres des distributions de probabilité du MMC.
2. Apprendre la structure (topologie du graphe) du MMC.
3. Réaliser l'apprentissage de façon incrémentale (une seule observation à la fois).
4. Réaliser l'apprentissage et l'inférence en temps réel (24 Hz pour une caméra vidéo).

Ces problèmes sont difficiles à résoudre ensemble, car, malgré l'existence des techniques pour l'apprentissage des paramètres et de la structure, il n'existe pas de technique capable de faire les deux à la fois, de façon incrémentale et en temps réel.

1.3 Contributions

La contribution principale de cette thèse est une technique de type “apprendre et prédire” basée sur une extension des MMC, proposée par nous. Cette extension reçoit le nom de “Modèles de Markov Cachés Grandissants” (MMCG) et sa particularité principale est que les paramètres et le nombre d’états ne sont pas fixés sinon qu’ils évoluent continuellement, au fur et à mesure que plus d’observations sont disponibles. Même si les MMCG ont été formulés dans le contexte de la prédiction des mouvements, il faut noter qu’ils ont été conçus comme une approche générale, pouvant être appliqués dans tous les cas ou les MMC conventionnels sont utilisés comme une approximation discrète pour modéliser un processus continu.

Pour pouvoir appliquer les MMCG à la prédiction des mouvements, on a pris la décision de modéliser les mouvements en utilisant un espace augmenté, qui ajoute l’état que l’objet prétend atteindre aux autres variables d’état classiques telles que la position et la vitesse.

En utilisant ensemble les MMCG et l’état augmenté, on a développé une nouvelle technique de prédiction des mouvements, ayant les avantages suivants par rapport aux autres techniques basées sur MMC dans la littérature :

1. Les paramètres du modèle aussi bien que ça structure sont estimés en utilisant un algorithme incrémental.

2. L’algorithme d’apprentissage est non supervisé et il est défini en termes des paramètres intuitifs.

3. La structure apprise ne consiste pas d’un ensemble de “trajectoires typiques” isolées, mais elle représente les comportements en termes de buts, ou destinations qu’un objet prétend atteindre.

4. Même si la structure apprise est plus riche que dans d’autres approches, elle reste assez simple pour permettre l’utilisation d’algorithmes exacts d’inférence.

Chapitre 2 : Modèles probabilistes

Ce chapitre est une introduction générale aux concepts des probabilités et des modèles probabilistes.

Notre travail se place dans le contexte des théories du raisonnement plausible développés par Jaynes, elles constituent une extension de la logique classique introduisant les probabilités comme un mécanisme d’inférence.

2.2 De la logique aux probabilités

2.2.1 Propositions logiques

On va travailler sur des propositions logiques pouvant être vraies ou fausses. Ces propositions peuvent être manipulées en utilisant des opérateurs booléens :

- L'opérateur *AND*, dénoté par un espace ou, dans le cas des variables indexées, par des indices séparés par deux points.
- L'opérateur *OR*, qui sera représenté avec le symbole $+$.
- L'opérateur *NOT*, indiqué par le symbole $\neg$.

2.2.2 Probabilité d'une proposition

Parfois, il n'est pas possible de déterminer de façon concluante si une proposition $\mathcal{A}$ est vraie ou fausse, mais on peut avoir des raisons pour croire que l'une de ces valeurs est plus vraisemblable que l'autre. Nous exprimons ce genre de connaissance comme la probabilité conditionnelle de la valeur de $\mathcal{A}$ sachant nos connaissances préalables π.

Règles quantitatives pour les propositions. Les règles quantitatives permettent de manipuler les probabilités pour réaliser l'inférence, de façon semblable à ce qu'on fait avec les opérateurs booléens.

Règle du produit

$$P(\mathcal{A}\ \mathcal{B}) = P(\mathcal{A})P(\mathcal{B} \mid \mathcal{A}) = P(\mathcal{B})P(\mathcal{A} \mid \mathcal{B}) \tag{1}$$

Règle de la normalisation

$$P(\mathcal{A}) + P(\neg\mathcal{A}) = 1 \tag{2}$$

Règle de l'addition

$$P(\mathcal{A} + \mathcal{B} \mid \mathcal{C}) = P(\mathcal{A} \mid \mathcal{C}) + P(\mathcal{B} \mid \mathcal{C}) - P(\mathcal{A}\ \mathcal{B} \mid \mathcal{C}) \tag{3}$$

2.2.3 Variables

Jusqu'ici, nous avons seulement parlé des propositions logiques, mais, fréquemment, nous voulons aussi raisonner en termes de variables – appelées aussi variables aléatoires – qui représentent les facteurs relevants d'un problème donné. Une variable discrète V a un domaine associe D_v, qui est l'ensemble des valeurs que cette variable peut prendre.

Une fois les variables définies, il est possible de les utiliser pour formuler des propositions logiques. Dans ce document, ces propositions seront indiquées en les entourant par des crochets, ainsi $[V = 1]$ dénote la proposition logique "la valeur de V est 1".

Règles quantitatives pour les variables. Comme pour les propositions, il est aussi possible de manipuler les variables à l'aide des règles quantitatives. Étant donné qu'il y a des différences entre les variables à domaine continu et les variables à domaine discret, on définit les règles selon le cas.

- *Variables discrètes*

– Règle du produit.

$$P(A\,B) = P(A)P(B \mid A) = P(B)P(A \mid B) \tag{4}$$

– Règle de la normalisation.

$$\sum_A P(A) = 1 \tag{5}$$

- *Variables continues*

– Règle du produit.

$$P(A\,B) = P(A)P(B \mid A) = P(B)P(A \mid B) \tag{6}$$

– Règle de la normalisation.

$$\int_{-\infty}^{\infty} g(A)dA = 1 \tag{7}$$

Autres identités utiles.

Règle de la marginalisation

$$\sum_B P(A\,B) = \sum_B P(B)P(A \mid B) = P(A) \qquad \text{(cas discrét)} \tag{8}$$

$$\int_{-\infty}^{\infty} P(A\,B)dB = \int_{-\infty}^{\infty} P(B)P(A \mid B)dB = P(A) \qquad \text{(cas continu)} \tag{9}$$

Règle de Bayes

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} = \frac{P(B \mid A)P(A)}{\sum_A P(B \mid A)P(A)} \qquad \text{(cas discrét)} \tag{10}$$

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} = \frac{P(B \mid A)P(A)}{\int_{-\infty}^{\infty} P(B \mid A)P(A)dA} \qquad \text{(cas continu)} \tag{11}$$

2.2.4 Décomposition de la probabilité conjointe

L'utilisation des règles quantitatives n'est pas limitée au cas de deux variables : du fait que le produit de deux variables aléatoires est aussi une variable aléatoire, les règles sont applicables à un nombre arbitraire de variables.

Les dépendances entre les variables sont dénotées formellement par leur probabilité conjointe, et la façon ou elle se décompose comme un produit de probabilités plus simples grâce à l'utilisation de la règle du produit.

Après avoir choisi une décomposition, il est possible de simplifier encore plus la probabilité conjointe sur la base des hypothèses d'indépendance conditionnelle. Ces hypothèses s'appliquent quand nous considérons qu'une variable $V1$ ne proportionne pas d'informations additionnelles par rapport à une variable $V2$ si la valeur d'une troisième variable $V3$ est connue.

2.2.5 Inférence

Les règles que nous avons décrites peuvent être utilisées pour définir un modèle qui décrit un phénomène ou processus quelconque ; un tel modèle probabiliste doit être spécifié en énumérant les variables qui le composent et ces domaines correspondants, ainsi que la décomposition de la conjointe utilisée.

L'application principale des modèles probabilistes est l'inférence : trouver les valeurs des variables inconnues en fonction des variables dont on connaît la valeur – l'évidence – à travers de l'application de la règle de Bayes.

Étant donné que l'inférence bayésienne est NP-difficile, il est nécessaire de trouver des solutions pour réduire la complexité. Cela peut se faire, par exemple, en appliquant des hypothèses d'indépendance conditionnelle, ou des algorithmes approximatifs d'inférence.

2.2.6 Formes paramétriques

Jusqu'ici, nous n'avons pas expliqué comment les probabilités qui conforment la conjointe son définies. Dans ce document, ces probabilités seront choisies parmi quelques distributions élémentaires qui s'expriment en fonction d'un certain nombre de paramètres et qu'on appelle donc "formes paramétriques".

- Distribution uniforme

$$P([V = v_i]) = \mathbf{U}_V(v_i) = \frac{1}{|D_V|}, \forall v_i \in D_V \tag{12}$$

- Tableaux de probabilité conditionnelle

$$P([A = i] \mid [B = j]\ [C = k]) = T_{i,j,k}$$

- Distribution gaussienne

$$P([V = v_i]) = \mathbf{G}(v_i;\ \mu,\ \Sigma) \tag{13}$$

$$= |2\pi\Sigma|^{-1/2} \exp\left[-\frac{1}{2}(V-\mu)^T\Sigma^{-1}(V-\mu)\right] \tag{14}$$

2.2.7 Apprentissage

Après la définition du modèle probabiliste, il est nécessaire d'assigner des valeurs aux paramètres de chaque distribution élémentaire dans la décomposition. Même si cela peut être fait à la main, il est aussi possible d'apprendre (estimer) les valeurs des paramètres à partir des données expérimentales.

Le cas le plus simple est celui des variables discrètes ou les probabilités sont calculées en estimant la fréquence avec laquelle les différentes valeurs des variables apparaissent dans les données. Pour cela, on peut utiliser des techniques telles que les a priori de Dirichlet ou la loi de succession de Laplace.

Une situation plus difficile, est quand les données sont bruitées et il est, donc, impossible de connaître avec certitude la vraie valeur des variables. Dans ce cas, la solution la plus utilisée consiste à appliquer l'algorithme Expectation-Maximization qui consiste basiquement à utiliser des expectations sur le nombre de fois qu'une valeur quelconque a été observée dans les données d'apprentissage.

2.3 Le filtre de Bayes

Cette section introduit le filtre de Bayes, un outil probabiliste qui est à la base des approches décrites dans les chapitres 4 et 5. L'objective du filtre de Bayes est de calculer une estimation probabiliste de l'état actuel d'un système dynamique - qui n'est pas directement observable, donc "caché" – à partir d'une séquence d'observations.

2.3.1 Modèle probabiliste

Variables. Le filtre de Bayes s'exprime en fonction de deux types de variables :

S_t L'état du système au moment t.

O_t L'observation obtenue au moment t.

Étant un modèle abstrait, le filtre de Bayes ne fait aucune hypothèse quant à la nature discrète ou continue des variables d'état et d'observation. Ces hypothèses sont faites pour des spécialisations du filtre de Bayes telles que le filtre de Kalman ou les Modèles de Markov Cachés.

Décomposition. La probabilité conjointe pour le filtre de Bayes est définie sur la base de deux hypothèses d'indépendance conditionnelle :

1. Sachant l'état, les observations sont indépendantes les unes des autres:

$$P(O_t \mid O_{1:t-1}\, S_{1:t}) = P(O_t \mid S_t) \tag{15}$$

2. Sachant l'état précédent, les états antérieurs n'apportent aucune connaissancee additionnelle par rapport à l'état actuel:

$$P(S_t \mid S_{1:t-1}) = \begin{cases} P(S_1) & \textit{for } t = 1 \\ P(S_t \mid S_{t-1}) & \textit{otherwise} \end{cases} \tag{16}$$

Et la distribution conjointe est :

$$P(S_{1:T}\; O_{1:T}) = P(S_1)P(O_1 \mid S_1)\prod_{t=1}^{T} P(S_t \mid S_{t-1})P(O_t \mid S_t) \tag{17}$$

2.3.2 Formes paramétriques

Le filtre de Bayes ne définit aucune forme paramétrique et, en conséquence, aucun mécanisme d'apprentissage.

2.3.3 Inférence

L'une des principales utilisations des filtres de Bayes est de répondre à la question probabiliste $P(S_{t+H} \mid O_{1:t})$. Le cas le plus commun est le filtrage ($H = 0$), mais la prédiction ($H > 0$) et le lissage ($H < 0$) sont aussi des opérations réalisées fréquemment.

Le filtre de Bayes a une propriété que contribue énormément à sa popularité, le filtrage peut s'effectuer de façon très efficace en utilisant l'expression suivante :

$$P(S_t \mid O_{1:t}) = \frac{1}{Z} P(O_t \mid S_t) \sum_{S_{t-1}} [P(S_t \mid S_{t-1}) P(S_{t-1} \mid O_{1:t-1})] \tag{18}$$

Si nous définissons récursivement $P(S_{t-1}) = P(S_{t-1} \mid O_{1:t-1})$, il est possible de décrire un filtre de Bayes avec seulement trois variables, ce qui donne l'expression suivante.

$$P(S_{t-1}\ S_t\ O_t) = P(S_{t-1}) P(S_t \mid S_{t-1}) P(O_t \mid S_t) \tag{19}$$

2.3.4 Spécialisations du filtre de Bayes

Il existe plusieurs spécialisations du filtre de Bayes, trois des plus populaires sont : le filtre de Kalman (variables continues), les Modèles de Markov Cachés (variables d'état discrètes), et le filtre à particules (approximation par echantillonage).

Chapitre 3 : Prédiction des mouvements intentionnels

Les approches basiques de prédiction de mouvements sont basées sur ce que Dennett appelle "l'attitude physique" envers l'objet : ils essayent d'expliquer et prédire le comportement des objets en termes de ses propriétés physiques et des lois de la physique. Néanmoins, une telle approche ne peut être appliqué à des objets qu'on peut considérer comme "rationnels" tels que des piétons ou des véhicules, dans ce cas, le comportement peut être mieux expliqué en prenant "l'attitude intentionnelle" envers l'objet, qui consiste à essayer de reproduire le processus de raisonnement qui détermine les actions de l'objet. Malheureusement, l'automatisation d'un tel processus est au-delà de nos capacités actuelles.

Récemment, une nouvelle famille d'approches situées à mi-chemin entre les deux "attitudes" a émergé. L'idée de base de ces approches est que, dans un environnement donné, les comportements des objets peuvent être observés de façon consistante, donc il suffit d'observer ces comportements pour les apprendre et ensuite les utiliser pour prédire.

Dans ce chapitre, nous examinons les approches de cette dernière famille, en les décomposant en trois catégories :

1. Trajectoires prototype.

2. Modèles d'état discrets.

3. Autres représentations.

En particulier, nous nous intéressons à étudier comment les approches de chaqu'une de ces catégories résolvent trois problèmes :

1. Représentation.

2. Apprentissage.

3. Prédiction.

3.2 Une note à propos de la sémantique

Dans ce document, nous allons faire une distinction entre un "comportement" et un "mouvement" typiques. Le premier décrit ce que l'objet est en train d'effectuer en prenant l'attitude intentionnelle envers lui (ex. l'objet va au toilette), pendant que le deuxième consiste en une description mathématique des mouvements effectués (ex. une série de positions dans le monde).

3.3 Trajectoires prototype.

Les approches de cette catégorie cherchent à réunir des trajectoires similaires en groupes (clusters) qui correspondent à des mouvements typiques. Après, pour chaqu'un de ces groupes, une seule trajectoire est calculée et utilisée pour représenter le groupe entier, et donc, le mouvement typique.

3.3.1 Représentation

Les trajectoires prototype sont souvent représentées comme des séquences de points dans l'espace continu des états. La plupart des approches ne modélisent pas le temps de façon explicite, et elles font l'hypothèse que les points de la séquence sont régulièrement distribués dans le temps. Parfois, une mesure de la "largeur" du groupe fait aussi partie de la représentation, par exemple en [Makris and Ellis, 2002, Junejo et al., 2004, Vasquez and Fraichard, 2004].

3.3.2 Apprentissage

Les trajectoires prototype sont obtenues à l'aide d'algorithmes classiques de clustering [voir Kaufman and Rousseeuw, 1989, Jain et al., 1999]. Il y a trois problèmes à résoudre : a) déterminer le nombre de groupes, b) trouver les groupes ; et c) construire les trajectoires prototype à partir des groupes. Nous avons identifié deux façons différentes de résoudre ces problèmes en fonction des algorithmes de clustering utilisés.

Algorithmes basés sur des modèles. Les algorithmes basés sur des modèles doivent leur nom au fait qu'ils ne représentent pas explicitement les groupes, mais, à la place, ils essayent de trouver un nombre réduit de modèles qui représentent au mieux les données par rapport à une mesure globale d'optimalité. Dans le cas des trajectoires, ces modèles correspondent directement aux trajectoires prototype dont nous avons parlé, donc elles sont calculées directement par l'algorithme de clustéring.

L'inconvénient le plus important de ces algorithmes est qu'ils nécessitent de connaître a priori le nombre de groupes qu'il faut trouver, ce qui présente le nouveau problème de l'estimation de ce nombre. La plupart des approches de prédiction supposent que ce nombre est connu [ex. Hu et al., 2004a]. Mais il existe certaines approches qui sont capables d'estimer ce nombre à partir d'une estimation initiale [ex. Bennewitz et al., 2002].

Algorithmes deux a deux. Les algorithmes deux à deux se basent dans l'utilisation d'une mesure de similarité (ex. distance euclidienne) qui est utilisée pour comparer des éléments deux à deux et décider s'ils appartiennent au même groupe. La sortie de ce processus consiste en ensembles d'éléments, ce que, dans le cas qui nos occupe, implique la nécessité de calculer la trajectoire prototype pour chaque groupe après la finalisation du clustering. D'un autre côté, ce type d'algorithmes a l'avantage de calculer de façon automatique le nombre de groupes.

Exemples de techniques de modélisation et prédiction de mouvements basées sur ce type d'algorithme sont [Makris and Ellis, 2002, Buzan et al., 2004, Junejo et al., 2004].

3.3.3 Prédiction

La prédiction pour cette catégorie d'approches consiste principalement à trouver la trajectoire prototype qui correspond au mieux à une séquence partiale d'observations ; et à utiliser cette trajectoire comme une prédiction du mouvement future.

Ces approches souffrent de deux inconvénients : a) seulement des trajectoires qui ont été observées peuvent être prédites ; et b) elles ont une représentation déterministe du temps et de l'évolution de l'état, ce qui réduit leur utilité pour prédire les états futurs de l'objet.

3.4 Modèles d'état discret

Ces approches se basent sur l'utilisation d'un modèle discret comme un outil approximatif d'analyse pour des mouvements continus. À la base de ces approches, on trouve les chaînes de Markov, qui modélisent le temps comme une variable discrète, et représentent l'espace avec un nombre fini d'états discrets.

Les chaînes de Markov, néanmoins sont relativement peu utilisées pour la prédiction des mouvements [ex. Tadokoro et al., 1995, J. Rittscher and Stein, 2003], les techniques préférées sont des dérivations du filtre de Bayes telles que les Modèles de Markov Cachés.

3.4.1 Représentation

Les mouvements typiques sont souvent représentés de deux façons différentes : a) en utilisant un seul MMC pour tous les comportements [Walter et al., 1999]; b) en utilisant un MMC différent

pour chaque comportement [Makris and Ellis, 2002, Bennewitz et al., 2002]. Dans les deux cas, les trajectoires typiques sont représentées en appliquant des contraintes à la structure des MMC, pour construire des graphes en forme des chaînes.

3.4.2 Apprentissage

Le problème de l'apprentissage pour ces approches se compose de deux sous-tâches : a) l'apprentissage de la structure ; et b) l'apprentissage des paramètres du modèle.

Apprentissage de la structure. Malgré l'existence de plusieurs algorithmes d'apprentissage de structure pour des modèles d'état discret [ex. Stolcke and Omohundro, 1993, Friedman, 1997, Brand, 1998], ils sont peu utilisés dans le contexte de l'apprentissage des mouvements typiques. Une exception notable est le travail de [Brand and Kettnaker, 2000]. À la place, la structure est, soit fixée à priori, soit estimée en utilisant des mécanismes ad hoc.

Le plus populaire de ces mécanismes est l'utilisation des techniques d'apprentissage utilisées pour déterminer des trajectoires prototype, et ensuite transformer ces dernières dans un modèle d'état discret [ex. Bennewitz et al., 2002, Koller-Meier and Van Gool, 2001, Makris and Ellis, 2002]. D'autres approches incluent des techniques hiérarchiques [Minnen and Wren, 2004] ou l'utilisation d'une carte de l'environnement [Liao et al., 2003].

Apprentissage des paramètres. L'approche la plus répandue pour l'apprentissage des paramètres est l'utilisation de l'algorithme de Baum-Welch [ex. Makris and Ellis, 2002, Liao et al., 2003]. Autres auteurs fixent les paramètres par rapport à leurs connaissances sur la façon dans laquelle les objets se deplacent [Bennewitz et al., 2005].

3.4.3 Prédiction

Dans cette catégorie d'approches, les mouvements sont prédits en utilisant l'inférence bayésienne. On peut aussi bien trouver des approches qu'utilisent l'inférence exacte [Brand and Kettnaker, 2000] que des techniques approximatives telles que les filtres à particules [Walter et al., 1999, Koller-Meier and Van Gool, 2001].

3.5 Autres Approches

3.5.1 Réseaux de neurones

On trouve aussi des approches qui se basent sur l'utilisation de plusieurs couches de réseaux de neurones pour représenter les mouvements typiques. Fréquemment [ex. Johnson and Hogg, 1995, Sumpter and Bulpitt, 2000], on trouve trois couches avec les fonctions suivantes : a) discrétisation de l'espace ; b) description des trajectoires ; c) classification des comportements. Autres auteurs [Hu et al., 2004b] proposent des diverses modifications a ce modèle de base.

3.5.2 Approches orientées aux buts

Ces approches [Dee and Hogg, 2004, Foka and Trahanias, 2002, Bruce and Gordon, 2004] se caractérisent pour représenter les comportements en termes des endroits que les objets prétendent atteindre “les buts”. Donc, ils doivent apprendre ces buts à partir des données disponibles. Une fois les buts appris, les trajectoires sont prédites en utilisant un algorithme qui calcule le chemin que l’objet va parcourir pour arriver à son but à partir de sa position actuelle.

3.5.3 Autres

Une approche différente a été proposée par [Kruse and Wahl, 1998]. Ils modélisent les mouvements en termes de sous trajectoires qui s’enchaînent de manière probabiliste pour former des trajectoires typiques.

Chapitre 4 : Modèles de Markov Cachés

Ce chapitre est une introduction aux Modèles de Markov Cachés (MMC), qui constituent la base de cette thèse. On présente si bien la partie “classique” de la théorie, que le sujet moins standardisé de l’apprentissage de structure.

4.2 Modèle probabiliste

Les Modèles de Markov Cachés sont une spécialisation du filtre de Bayes pour des variables d’état discrètes et des variables d’observation discrètes et continues. Nous avons privilégié la discussion des variables d’observation continues parce qu’elles sont mieux adaptées à notre problème.

4.2.1 Variables

Comme pour la version récursive du filtre de Bayes, un MMC peut être défini en fonction de trois variables :

S_t, S_{t-1}. L’état actuel et l’état précèdent, qui sont des entiers dans l’intervalle $[1, N]$.

O_t. L’observation actuelle, qui est un vecteur dans $\mathbb{R}^M$.

4.2.2 Décomposition

La décomposition est la même que pour le filtre de Bayes :

$$P(S_{t-1}\ S_t\ O_t) = P(S_{t-1})P(S_t \mid S_{t-1})P(O_t \mid S_t) \tag{20}$$

Où la probabilité a priori pour l’état est calculée récursivement :

$$P(S_{t-1}) = P(S_{t-1} \mid O_{1:t-1}) \tag{21}$$

4.2.3 Formes paramétriques

Les MMC font une hypothèse d'indépendance conditionnelle en plus par rapport au filtre de Bayes : les probabilités d'observation et de transition sont considérées comme étant stationnaires, c'est-à-dire, indépendantes du temps.

$$P(O_i \mid S_i) = P(O_j \mid S_j) \quad \forall\, i,\ j \in \{1, \cdots, T\} \tag{22}$$
$$P(S_i \mid S_{i-1}) = P(S_j \mid S_{j-1}) \quad \forall\, i,\ j \in \{2, \cdots, T\} \tag{23}$$

Cette hypothèse permet de définir les formes paramétriques sans prendre en compte le temps :

- $P([S_1 = i]) = \pi_i$. L'a priori sur l'état.
- $P([S_t = j] \mid [S_{t-1} = i]) = a_{i,j}$. Les probabilités de transition.
- $P(O_t \mid [S_t = i]) = \mathbf{G}(O_t;\ \mu_i,\ \sigma_i)$. Les probabilités d'observation.

Nous allons dénoter l'ensemble des paramètres d'un MMC par: $\lambda = \{\pi, A, b\}$.

4.3 Inférence

Les tâches d'inférence peuvent être classifiées en deux types : a) l'inférence en ligne, qui s'actualise chaque fois qu'une nouvelle observation est disponible ; et b) l'inférence hors ligne qui traite des séquences entières d'observations.

4.3.1 Inférence en ligne

La clé pour l'inférence en ligne est la mise à jour de l'estimation de l'état. À partir de là, l'inférence se réalise en utilisant cette estimation comme a priori, et en appliquant l'inférence bayésienne.

Filtrage. Le nom du filtrage vient du fait qu'il filtre le bruit des estimations pour ainsi estimer l'état du système. Le filtrage se réalise en appliquant l'expression suivante :

$$P(S_t \mid O_{1:t}) = \frac{1}{Z} \underbrace{P(O_t \mid S_t)}_{update} \underbrace{\sum_{S_{t-1}} [P(S_t \mid S_{t-1}) P(S_{t-1} \mid O_{1:t-1})]}_{prediction} \tag{24}$$

La complexité du filtrage est $O(N^2)$.

Prédiction. Prédire l'état futur consiste, basiquement, en prendre l'estimation de l'état actuel et la propager dans le futur pour un nombre fini de pas de temps H, connu comme l'horizon temporel.

$$P(S_{t+H} \mid O_{1:t}) = \sum_{S_{t+H-1}} [P(S_{t+H} \mid S_{t+H-1}) P(S_{t+H-1} \mid O_{1:t})] \tag{25}$$

La complexité de la prédiction est $O(TN^2)$.

4.3.2 Inférence hors ligne

En vue du fait que l'inférence hors ligne travaille sur l'espace de toutes les séquences possible d'états ayant une longueur donnée T, il semblerait que le coût de l'inférence devrait avoir une complexité exponentielle. Heureusement, l'utilisation de techniques de programmation dynamique permet de réduire la complexité de l'inférence à $O(TN^2)$.

L'algorithme "forward-backward". L'idée de cet algorithme est d'utiliser des techniques de programmation dynamique pour éviter les computations redondantes liées à l'application répétitive de la règle de marginalisation pour réaliser l'inférence. Il calcule deux ensembles différents de variables qui peuvent ensuite être utilisés directement pour répondre des questions probabilistes. L'intérêt de l'utilisation de ces variables vient du fait que leurs algorithmes de calcul sont d'ordre $O(N^2)$.

Probabilité "forward"

$$\alpha_t(i) = \left[\sum_{j=1}^{N} \alpha_t(j) P([S_t = i] \mid [S_{t-1} = j]) \right] P(O_t \mid [S_t = i]) \tag{26}$$

Probabilité "backward"

$$\beta_t(i) = \sum_{j=1}^{N} P([S_{t+1} = j] \mid [S_t = i]) P(O_{t+1} \mid [S_{t+1} = j]) \beta_{t+1}(j) \tag{27}$$

Lissage. Le lissage est similaire au filtrage, mais il prend en compte la séquence entière d'observations pour estimer l'état pour un instant donné. Les estimations ainsi obtenues sont plus précises qu'avec le filtrage.

Le lissage se réalise en utilisant les probabilités "forward" et "backward" :

$$P([S_t = i] \mid O_{1:T}) = \frac{1}{p_O} \alpha_t(i) \beta_t(i) \tag{28}$$

Proche du lissage est le calcul de la probabilité d'une transition connaissant une séquence d'observations :

$$P([S_{t-1} = i]\ [S_t = j] \mid O_{1:T}) = \frac{\alpha_{t-1}(i) P([S_t = j] \mid [S_{t-1} = j]) P(O_t \mid [S_t = j]) \beta_t(j)}{p_O} \tag{29}$$

L'algorithme de Viterbi. L'algorithme de Viterbi répond à la question "quelle est la séquence d'états qui correspond avec la plus grande probabilité à une séquence d'observations donnée?" Essentiellement, l'algorithme de Viterbi est identique au calcul des probabilités "forward", mais en remplaçant l'addition par un opérateur de maximisation :

$$\delta_t(j) = \max_i [\delta_{t-1}(i) P([S_t = j] \mid [S_{t-1} = i])] P(O_t \mid [S_t = j]) \tag{30}$$

Ou $\delta_t(j)$ représente la vraisemblance maximum d'observer la séquence partielle d'observations $O_{1:t}$ et d'être dans l'état j à l'instant t. L'algorithme garde aussi, pour chaque instant t et chaque état j, l'état précèdent $\psi_t(j)$ qui mène a j avec probabilité maximum.

Classification. La classification consiste à choisir parmi plusieurs MMC, celui qui correspond le mieux à une séquence d'observations donnée.

4.3.3 Stabilité numérique

A l'heure d'implémenter des MMC dans des ordinateurs, il est fréquent de trouver des problèmes de stabilité numérique pendant l'exécution des algorithmes de Viterbi et "forward-backward". Ces problèmes viennent des multiplications de longues séries de valeurs plus petites que 1, ce qui vite dépasse la capacité de représentation de la machine. La solution à ce problème dépend de l'algorithme en question :

Algorithme de Viterbi: utiliser des logarithmes des probabilités et des sommations a la place des probabilités et des multiplications.

Algorithme "forward-backward": multiplier les probabilités pour chaque pas de temps par un facteur d'échelle, calculé comme :

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{31}$$

4.4 Apprentissage des paramètres

La technique standard pour réaliser l'apprentissage des paramètres d'un MMC est l'algorithme Baum-Welch.

4.4.1 L'algorithme de Baum-Welch

Cette section décrit l'algorithme de Baum-Welch, qui est détaillé dans le chapitre correspondant. Le principe de cet algorithme est l'utilisation des probabilités forward et backward pour re-estimer les paramètres du modèle à partir d'une estimation précédente. L'algorithme Baum-Welch est, en fait, une spécialisation de l'algorithme Expectation-Maximization.

4.4.2 Algorithmes incrémentaux

Dans cette section nous survolons les deux variantes incrémentales de l'algorithme Baum-Welch, proposées par [Neal and Hinton, 1998] et par [Singer and Warmuth, 1996].

4.5 Structure des transitions

Jusqu'à maintenant, nous avons considéré des modèles complètement connectés ou ergodiques, oú chaque état peut être atteint depuis n'importe quel autre état dans un seul pas du temps. Toutefois, dans certaines applications, il est mieux d'appliquer des contraintes en interdisant certaines de ces transitions ; quand on fait cela, on dit que l'on fixe la structure, ou la topologie du MMC. Dans la pratique, le choix d'une topologie se réalise en fixant à zéro la valeur de certains éléments de la matrice de transition.

Une situation intéressante arrive quand la matrice de transition se compose principalement de zéros (matrice creuse). Dans ce cas, il est possible d'utiliser des représentations plus performantes qu'une matrice, si bien en termes de stockage qu'en temps de traitement.

En plus d'améliorer la performance, le choix d'une structure a aussi des répercussions sur la qualité de l'inférence [Brand, 1998, Freitag and McCallum, 2000, Binsztok and Artières, 2005] et accélère l'apprentissage en réduisant le nombre de paramètres à être appris.

4.6 Apprentissage de la structure

Dans cette section, nous décrivons de manière sommaire les différentes familles d'approches pour faire l'apprentissage de structures.

4.6.1 Algorithmes de recherche locale

Ces approches [Friedman, 1997] partent d'une structure relativement simple, et procèdent en ajoutant des noeuds ou des arêtes jusqu'à l'obtention d'un modèle "optimale" par rapport à une critère donné.

4.6.2 Algorithmes « state merging »

Les algorithmes "state merging" [Stolcke and Omohundro, 1994, Seymore et al., 1999] fonctionnent dans le sens inverse des algorithmes précédents : ils partent d'une structure complexe et, après, ils la simplifient en fusionnant des états. À chaque itération de l'algorithme, la meilleure fusion est déterminée et le processus continue ainsi de suite, jusqu'à l'obtention d'un modèle "optimale".

4.6.3 Autres Algorithmes

On trouve aussi d'autres approches dans la littérature, telles que [Brand, 1998, Vasko et al., 1997, Lockwood and Blanchet, 1993, Freitag and McCallum, 2000]. Nous les décrivons avec plus de détail dans le chapitre correspondant.

Chapitre 5 : Modèles de Markov Cachés Grandissants

On a montré dans la première partie de cette thèse, que les Modèles de Markov Cachés constituent un puissant outil probabiliste. Néanmoins, pour pouvoir les appliquer à notre problème, il est nécessaire de disposer d'algorithmes incrémentaux d'apprentissage de la structure et des paramètres qui soient capables de fonctionner en temps réel.

Ce chapitre introduit la solution que nous proposons, qui est aussi la principale contribution de cette thèse : les Modèles de Markov Cachés Grandissants. Ils peuvent être décrits comme des MMC qui évoluent au cours du temps, et dont le nombre d'états, la topologie, et les paramètres des distributions de probabilité sont mises à jour chaque fois qu'une séquence d'observations est disponible.

Nous supposons que l'espace ou les objets évoluent a été discrétisé dans un nombre finit de régions discrètes, et chaqu'une de ces régions est représentée par un état discret dans le MMC.

L'intuition de base derrière notre approche est que la structure du MMC doit ressembler à celle de l'environnement : les transitions sont permises seulement quand les régions correspondantes sont voisines. Donc, l'apprentissage de la structure consiste à estimer la meilleure discrétisation possible de l'espace et en identifier les régions voisines. Nous avons attaqué ce problème en construisant une carte topologique de l'environnement. Pour l'apprentissage des paramètres, nous nous sommes basés sur l'approche proposée par [Neal and Hinton, 1998] et nous l'avons adapté pour gérer les changements dans la cardinalité des états et les observations continues.

5.2 La carte topologique

La carte topologique est une représentation discrète de l'espace des états qui prend la forme d'un graphe, dont les noeuds représentent des régions discrètes et les arêtes indiquent que les régions correspondantes sont contiguës : il est possible de se déplacer de façon continue d'une région à l'autre sans passer à travers d'aucune autre région.

Même si l'idée de carte topologique peut être facilement appliquée à les grilles régulières, il est généralement accepté que des représentations dites 'dynamiques' ou 'adaptatives' - qui discrétisent l'espace de façon irrégulière pour représenter au mieux un ensemble de données - sont plus performantes en termes de ressources.

Pour notre travail, nous avons choisi d'utiliser une famille d'approches - les réseaux topologiques [Martinetz and Schulten, 1991]- qui permettent d'obtenir des cartes topologiques en utilisant des algorithmes d'apprentissage incrémentaux.

5.3 Quantisation vectorielle et réseaux topologiques

À la base, les réseaux topologiques peuvent être vus comme des algorithmes pour la quantisation vectorielle. Cette dernière consiste à représenter une variété D-dimensionnelle continue en utilisant un ensemble fini de vecteurs de référence $D-$dimensionnels. Un point x de la variété est représenté en utilisant le vecteur de référence le plus proche c, par rapport à une mesure de distance donnée $d(x,y)$.

Le but de la quantisation vectorielle est de minimiser la distance moyenne, ou distorsion, entre les points qui appartiennent à la variété et les vecteurs de référence correspondants.

$$E = \sum_{i=1}^{K} \int_{x \in \mathcal{V}_i} d(x, c_i) P(x) dx \tag{32}$$

Cela peut se faire aussi en prenant des échantillons appartenant à la variété :

$$\hat{E} = \frac{1}{|X|} \sum_{i=1}^{K} \sum_{x_j \in \mathcal{V}_i} d(x_j, c_i) \tag{33}$$

La plus populaire des approches de quantisation vectorielle est l'algorithme k-means [Lloyd, 1957, Linde et al., 1980], dont il existe aussi une version incrémentale [MacQueen, 1967]. D'un autre côté, cet algorithme à deux problèmes importants : a) le nombre de vecteurs de référence doit être connu a priori ; et b) il est très sensible à l'initialisation.

Un des algorithmes alternatifs sont les réseaux auto organisés de Kohonen [Kohonen, 1995], la principale innovation de cette approche est que, quand un vecteur d'entrée (échantillon de la variété) est traité, seulement le vecteur de référence le plus proche et ses voisins sont mis à jour. Ces voisins sont indiqués de façon explicite, comme des liens ou des arêtes qui forment un réseau, et définissent une topologie.

5.3.1 Réseaux topologiques

Les réseaux topologiques développent l'idée des réseaux de Kohonen en apportant deux nouvelles capacités : a) des vecteurs de référence (noeuds) peuvent être crées pendant l'apprentissage ; et b) la structure du réseau est aussi apprise en ajoutant/éliminant des arêtes dans le réseau.

Parmi les divers réseaux topologiques existants [Martinetz and Schulten, 1991, Fritzke, 1995, Marsland et al., 2002], nous avons choisi la carte topologique instantanée de Jockusch et Ritter.

5.4 La carte topologique instantanée

La carte topologique instantanée (CTI) [Jockusch and Ritter, 1999] présente deux avantages par rapport aux autres réseaux topologiques : a) elle est capable de gérer des données corrélées dans le temps ; et b) il a un nombre réduit de paramètres ayant une claire interprétation physique ; en plus, il ne demande pas des connaissances a priori par rapport à la topologie ou la taille de la variété - dans notre cas, l'espace où les objets se deplacent - qui doit être apprise.

L'algorithme se base dans l'utilisation d'une mesure de distance, qui, dans notre cas, est celle de Mahalanobis.

5.4.1 Définitions

L'algorithme construit de façon incrémentale un ensemble de noeuds, et un ensemble d'arêtes qui connectent ces noeuds. L'entrée de l'algorithme consiste en des vecteurs d'entrée qui, dans cette thèse, vont être identifiés aux observations fournies par les capteurs.

Associé à chaque noeud, il y a un vecteur de poids.

L'ensemble des autres noeuds auxquels un noeud donné est connecté par des arêtes est appelé son voisinage.

5.4.2 Algorithme

La CTI a seulement trois paramètres :

La matrice de covariance (Σ). Utilisée pour calculer la distance de Mahalanobis.

Le seuil d'insertion (τ). Définisse la distance moyenne entre noeuds.

Le taux de lissage (ε). Régule la vitesse d'adaptation des vecteurs de référence..

L'algorithme est détaillé dans le chapitre correspondant. Il peut être décomposé en quatre pas :

Appariement. Trouve les deux noeuds les plus proches au vecteur d'entrée.

Adaptation des poids. Le noeud le plus proche est déplacé vers le vecteur d'entrée d'accord au facteur de lissage choisi

Adaptation des arêtes. Créé une arête entre les deux noeuds les plus proches s'il n'existait pas auparavant. Il peut éventuellement effacer d'autres arêtes s'ils sont redondants.

Adaptation des noeuds. Créé des nouveaux noeuds (et les arêtes correspondants) si le vecteur d'entrée est trop éloigné du reste du réseau. Il peut aussi effacer des arêtes s'ils sont redondants.

5.4.3 Propriétés

Convergence. La CTI, n'a pas des propriétés strictes de convergence vers un minimum local de la distorsion. Il est possible, par contre, de montrer que :

$$\hat{E} \leq \tau \tag{34}$$

Nombre d'arêtes. [Dwyer, 1989] a montré que dans la plupart des cas, le nombre d'arêtes pour une CTI dépende de façon linéaire du nombre de noeuds dans la carte, et que cela est indépendant de la dimension de l'espace en question.

Complexité. La CTI a été conçue pour l'apprentissage incrémental, la complexité temporelle de l'algorithme de mise à jour est de $O(N)$, et cela peut être réduit en utilisant des techniques hiérarchiques d'indexation de l'espace comme les R-Trees et ses extensions [Guttman, 1984, Beckmann et al., 1990].

5.5 Modèle probabiliste

Dans cette section, nous expliquons comment intégrer la CTI pour apprendre la structure d'un MMC. Nous présentons aussi la façon ou nous avons modifié l'apprentissage des paramètres pour prendre en compte une structure évolutive. Comme pour les MMC on va décrire notre approche comme un modèle probabiliste.

5.5.1 Variables

La seule différence a ce niveau entre les MMC et les MMCG est que le domaine de la variable d'état change à mesure que le temps avance. Mise à part cela, les deux modèles utilisent les mêmes variables.

- S_t, S_{t-1}, L'état actuel et l'état précèdent, qui sont des entiers dans l'intervalle $[1, S_k]$, ou S_k est le nombre d'état dans la structure après k pas de temps.
- O_t, L'observation actuelle, qui est un vecteur dans $\mathbb{R}^D$.

5.5.2 Décomposition

La décomposition est aussi la même que pour les MMC :

$$P(S_{t-1}\ S_t\ O_t) = P(S_{t-1})P(S_t \mid S_{t-1})P(O_t \mid S_t) \tag{35}$$

5.5.3 Formes paramétriques

Les formes paramétriques des MMCG sont aussi fondamentalement les mêmes que pour les MMC, mais la manière ou ces paramètres sont stockées est différente.

- $P(S_t)$. L'a priori sur l'état est aussi une multinomiale, mais à la place de stocker les probabilités directement dans un vecteur nous stockons les SSE (voir sec. 4.4) pour calculer ces probabilités. Les probabilités seront calculées directement sur ces valeurs.:

$$P(S_t = i) = \frac{\pi_i}{\sum_{S_t} \pi_{S_t}} \tag{36}$$

- $P(O_t \mid S_t)$. À différence des MMC toutes les gaussiennes des probabilités d'observation vont avoir la même covariance Σ:

$$P(O_t \mid S_t = i) = \mathbf{G}(O_t;\ \mu_i,\ \Sigma) \tag{37}$$

- $P(S_t \mid S_{t-1})$. Comme pour l'a priori, nous allons stocker des SSE dans la matrice de transition :

$$P(S_t = j \mid S_{t-1} = i) = \frac{a_{i,j}}{\sum_{S_t, S_{t-1}} a_{S_{t-1}, S_t}} \tag{38}$$

5.6 Inférence

L'inférence dans les MMCG est exactement la même que pour les MMC.

5.7 Apprentissage de la structure et des paramètres

L'algorithme d'apprentissage est la particularité la plus importante des MMCG. L'algorithme alterne entre deux activités, la mise à jour de la carte topologique, et l'estimation des paramètres des distributions de probabilité.

L'algorithme, détaillé dans le chapitre correspondant, a les paramètres suivants :

Valeur par défaut de l'a priori sur les états (π_0). Utilisé pour initialiser l'a priori quand des nouveaux états sont créés.

Valeur par défaut des transitions (a_0). Utilisé pour initialiser les probabilités de transition quand des nouveaux arêtes sont créés.

Matrice de covariance (Σ). Elle est utilisée pour calculer la distance de Mahalanobis dans la CTI et pour toutes les gaussiennes des probabilités d'observation.

Seuil d'insertion, et taux de lissage (τ), (ε). Utilisées par la CTI.

L'algorithme, qu'utilise des séquences complètes d'observations comme entrées, se décompose en trois parties :

Mise à jour de la carte topologique. En utilisant toutes les observations de la séquence.

Mise à jour de la structure du MMCG. Des états et transitions sont ajoutés ou effacés pour refléter la structure de la CTI.

Mise à jour des paramètres. Applique une version incrémentale de l'algorithme Baum-Welch.

Chapitre 6 : Apprentissage et prédiction des mouvements avec MMCG

Ce chapitre se concentre sur l'application des MMCG à la prédiction des mouvements de piétons et de véhicules. Notre hypothèse de base est que les objets se déplacent en fonction de leur intention d'atteindre un état particulier (leur but). En conséquence, nous modélisons le mouvement de l'objet en fonction d'un vecteur d'état augmenté qui est composé de deux ensembles de variables décrivant son état actuel et celui qu'il prétend atteindre.

Une conséquence de ce choix de modélisation est que les mouvements typiques ne correspondent plus à des trajectoires, mais a l'ensemble des chemins qu'un objet peut emprunter pour arriver à un but donné.

6.2 Modèle probabiliste

Jusqu'à maintenant, nous avons supposé que les observations nécessaires à l'inférence et l'apprentissage sont fournies par les capteurs à chaque pas de temps. Dans notre application, cette hypothèse n'est pas toujours vraie, au moins dans le cas des buts. En effet, la position d'un objet est observable tout le temps, mais la destination, par définition, seulement peut être observée à la fin d'une trajectoire.

Cette section décrit comment utiliser des MMCG pour créer des modèles de mouvement, et comment gérer les observations "manquantes" qui correspondent aux buts.

6.2.1 Variables

Nous allons considérer notre modèle probabiliste à deux niveaux. Dans le niveau le plus général, notre modèle se comporte comme un MMCG ordinaire, et l'espace augmenté n'est pas différent d'une autre définition quelconque d'état. Mais, dans un niveau plus bas, nous voulons faire la distinction entre l'état actuel et l'état but, donc nous allons décomposer les variables d'observation dans une composante actuelle O'_t et une composante but O''_t. Cela donne les variables suivantes :

- S_t, S_{t-1}, l'état augmenté actuel et l'état augmenté précédent, qui sont des entiers dans l'intervalle $[1, S_k]$, ou S_k est le nombre d'états dans la structure après k pas de temps..
- O_t, l'observation actuelle, qui est un vecteur dans $\mathbb{R}^{2D}$. Comme on a expliqué avant, les observations se décomposent en deux $O_t = [O'_t, O''_t]$.

6.2.2 Décomposition

Au plus haut niveau, la décomposition est la même que pour les MMCG :

$$P(S_{t-1}\ S_t\ O_t) = P(S_{t-1})P(S_t \mid S_{t-1})P(O_t \mid S_t) \tag{39}$$

Mais dans le niveau inférieur, les observations représentent l'occurrence conjointe de ses composantes : actuel et but.

$$P(O_t \mid S_t) = P(O'_t\ O''_t \mid S_t) \tag{40}$$

Nous faisons l'hypothèse que les deux composantes sont indépendantes si l'état actuel est connu :

$$P(O'_t\ O''_t \mid S_t) = P(O'_t \mid S_t)P(O''_t \mid S_t) \tag{41}$$

Donc, nous obtenons la décomposition suivante :

$$P(S_{t-1}\ S_t\ O'_t\ O''_t) = P(S_{t-1})P(S_t \mid S_{t-1})P(O'_t \mid S_t)P(O''_t \mid S_t) \tag{42}$$

6.2.3 Formes paramétriques

Les formes paramétriques sont les mêmes que pour les MMCG, mais dans le cas des probabilités d'observation on fixe des contraintes supplémentaires :

- $P(S_t)$. Pareil que pour les MMCG.
- $P(O_t \mid S_t)$. Grâce à notre hypothèse d'indépendance conditionnelle, nous pouvons écrire les probabilités d'observation comme un produit de probabilités : $P(O'_t\ O''_t \mid S_t) = P(O'_t \mid S_t)P(O''_t \mid S_t)$. Si nous définissons ces probabilités comme :

$$P(O'_t \mid [S_t = i]) = \mathbf{G}(O'_t;\ \mu'_i,\ \Sigma') \tag{43}$$

et:

$$P(O''_t \mid [S_t = i]) = \begin{cases} \mathbf{U}_{O''_t} & \text{if } O''_t \text{ is not available} \\ \mathbf{G}(O''_t;\ \mu''_i,\ \Sigma'') & \text{otherwise} \end{cases} \tag{44}$$

Et notant que $P(O_t \mid S_t)$ est soit un produit de gaussiennes, soit le produit d'une constante pour une gaussienne, nous pouvons réécrire cette probabilité comme une seule gaussienne :

$$P(O_t \mid [S_t = i]) = \frac{1}{Z}\mathbf{G}(O_t;\ \mu_i,\ \Sigma) \tag{45}$$

ou $\mu_i = [\mu'_i, \mu''_i]$, et Σ est une matrice diagonale en blocs :

$$\Sigma = \begin{bmatrix} \Sigma' & 0 \\ 0 & \Sigma'' \end{bmatrix} \tag{46}$$

et Z est une constante de normalisation.

- $P(S_t \mid S_{t-1})$. Pareil que pour les MMCG.

6.3 Inférence

L'inférence est la même que pour les MMC.

6.4 Apprentissage de la structure et des paramètres

L'apprentissage se réalise de façon standard en utilisant l'algorithme 7. Il est, cependant, nécessaire de prétraiter les séquences d'observations reçues pour ajouter la dernière observation de la séquence à chaque vecteur d'observation.

6.5 Exemple d'apprentissage : environnement unidimensionnel

6.6 Comparaison avec des approches existantes

Dans cette section nous comparons notre approche avec d'autres techniques existantes basées sur des modèles à états discrets, qui, en contraste avec nous, utilisent des trajectoires typiques pour définir la structure du modèle.

Redondance. Un premier problème de l'utilisation des trajectoires typiques vient du fait que les structures résultantes ne partagent pas des états. Un exemple de cette situation est montré dans la figure 6.6. Il y a deux trajectoires typiques différentes (fig. 6.6(a)) qui sont représentées dans la structure comme deux chaînes séparées (fig. 6.6(b)). Notre approche, par contre, produit un modèle plus compact (fig. 6.6(c)).

Combinaisons de comportements. Dans les approches basées sur des trajectoires typiques, les transitions entre comportements "différents" ne sont pas permises, en conséquence, le modèle n'est pas capable d'expliquer des mouvements constitués de "morceaux" des comportements déjà appris. Un exemple de ce problème est montré dans la fig. 6.7.

Sémantique. Un problème plus profond est lié à la sémantique des trajectoires typiques. Elles sont définies en termes de similarités, distances, ou d'autres mesures géométriques ou statistiques, sans jamais parler des causes qui sont à l'origine du mouvement. Dans d'autres mots, elles n'essaient pas de répondre à la question cruciale "pour quoi les trajectoires sont-elles typiques ?", question qui, de notre point de vue, faut répondre pour construire des vrais modèles génératifs du mouvement. D'un autre côté, nous sommes conscients que la modélisation des causes est très difficile dans le cas des êtres humains parce que cela implique se mettre à la place de la personne dont on est en train de prédire les mouvements. Nous considérons que, même si notre approche est loin d'être un modèle intentionnel satisfaisant, il se rapproche plus d'une explication causale des mouvements, grâce au fait qu'il est basé sur un modèle des intentions - même si ce modèle est grossier.

Chapitre 7: Plateforme expérimentale

Toutes les expériences qui ont été conduites dans cette thèse sont basées sur des ensembles des données qui ont été collectés dans trois environnements différents : le hall d'entrée de l'INRIA, le parking de l'INRIA et un parking dans l'université de Leeds. On a aussi utilisé des données synthétiques pour compléter nos expériences.

7.2 Hall d'entrée de l'INRIA

Le premier environnement que nous avons étudié est le hall d'entrée du bâtiment de l'INRIA Rhône-Alpes, qui est un environnement relativement ouvert qui contient des lieux "intéressants" tels qu'un bureau d'accueil, une cafétéria, des bornes d'information et plusieurs portes qui donnent accès aux différentes sections du laboratoire.

Pour recueillir les informations concernant aux piétons qui transitent dans cet environnement, on a utilisé un système de suivi visuel d'objets connecté à une caméra vidéo installée dans un des coins du hall. On a aussi développé un simulateur de cet environnement pour réaliser des expériences dans lesquelles nous connaissons les variables.

7.2.1 Le système de suivi

Le système de suivi que nous avons utilisé a été développé par l'équipe Prima du laboratoire GRAVIR. Ce système détecte et suit des objets mobiles dans un flux vidéo venant d'une caméra. Les informations proportionnées par le système sont la position et la taille des objets, dans le repère de l'image de la caméra. Ces informations sont en suite projetées dans le repère global, avant d'être utilisées dans notre algorithme. La chaîne complète de traitement est conformée par les éléments suivants :

1. Caméra et système de suivi. Produisent, à partir des images vidéo, des estimations de la position et de la taille des objets, exprimées dans le repère de la caméra.

2. Correction de la distorsion et projection homographique. Pour pouvoir projeter les informations dans le repère global, il faut d'abord éliminer la distorsion induite par l'utilisation des lentilles à grand angle. Après, la projection se réalise en utilisant une matrice d'homographie préalablement calculée.

3. Association des données. Pour améliorer les résultats, on a appliqué un filtre JPDA (Joint Probabilistic Data Association) sur les données projetées, de façon à minimiser le nombre de cas où le système de suivi "coupe" en plusieurs trajectoires les données correspondantes à un seul objet.

7.2.2 Le Simulateur

Le simulateur que nous avons développé est basé sur l'idée de "points de contrôle" qui représentent des lieux importants dans l'environnement tels que le bureau d'accueil ou les bornes d'information.

Basés sur ces points de contrôle, nous avons défini un certain nombre de trajectoires typiques, définies comme une liste des points de contrôle à traverser. Pour simuler une trajectoire, une trajectoire typique est sélectionnée au hasard et les observations correspondantes sont générées en utilisant un processus d'interpolation auquel on ajoute du bruit.

7.2.3 Les Données

Nous avons recueilli des données pendant une semaine en des moments différents de la journée. Le nombre des trajectoires que nous avons obtenues après avoir filtré celles qui étaient trop longues (plus de 250 observations) ou trop courtes (moins de 50) est de 2048.

Mis à part le filtrage que nous avons mentionné, les séquences d'observations récupérées n'ont subi aucun posttraitement. En conséquence, on compte plusieurs trajectoires "anomales" parmi cet ensemble de données. Ces trajectoires anomales correspondent à des erreurs de suivi, et elles ne représentent pas des mouvements réels (ou, au moins, des trajectoires complètes).

Ces problèmes ne sont pas présents dans les données synthétiques, que nous pouvons produire à souhait.

7.3 Le parking de l'université de Leeds

Le deuxième environnement que nous avons considéré est un parking dans l'université de Leeds. À différence du hall, on peut trouver deux types d'objets dans cet environnement (véhicules et piétons), une autre différence est que cet environnement est plus structuré, au moins dans le cas des voitures.

Les données ont été recueillies en utilisant un système de suivi différent de celui utilisé dans le hall, mais la différence la plus importante est que ces données ont été traitées manuellement pour corriger les erreurs de suivi. Approximativement 20% des trajectoires ont été altérées de cette façon dont certaines ont été complètement suivies à la main.

7.4 Le simulateur du parking de l'INRIA

En raison des difficultés que nous avons trouvées pour obtenir des données, nous avons décidé de continuer le développement de notre simulateur pour pouvoir ainsi disposer d'une plateforme expérimentale permettant de faire des expériences contrôlées et de travailler avec des environnements plus grands qu'avec les données réelles.

Nous avons donc produit un simulateur du parking de l'INRIA qui permet de simuler les mouvements des piétons et des véhicules en prenant en compte aussi des vitesses.

Chapitre 8 : Résultats expérimentaux

Nous avons réalisé des expériences extensives avec notre approche en utilisant les données décrites dans le chapitre 7. Nous nous sommes concentrés dans les questions suivantes :

1. Performance de l'apprentissage.
2. Exactitude des prédictions.
3. Fonctionnement en temps réel.
4. Généralité.

Ce chapitre présente les expériences que nous avons réalisées pour répondre à ces questions.

8.2 Exemples

Cette section présente informellement quelques exemples du fonctionnement de notre approche sans en discuter les détails techniques.

8.2.1 Le hall d'entrée

8.2.2 Le parking de Leeds

8.2.3 Le parking de l'INRIA

8.3 Résultats quantitatifs

Dans cette section, nous présentons les résultats que nous avons obtenus en étudiant nos questions.

8.3.1 Sélection des paramètres

Pour étudier la sensibilité de notre approche aux changements dans les paramètres, nous avons utilisé six ensembles de paramètres.

Dans tous les cas, nous avons utilisé des covariances sphériques pour les variables d'état.

Pour chaque ensemble de données, les valeurs ont été fixées de la façon suivante : le premier ensemble de paramètres est une estimation proposée par une personne. Les ensembles "Low CV" et "High CV" correspondent à des valeurs inferieures et superieures des covariances, respectivement ; les ensembles "Low IT" et "High IT" fonctionnent de façon analogue pour le coefficient d'insertion ; le dernier ensemble des paramètres est le meilleur que nous avons trouvé par rapport à sa parcimonie et exactitude, et, dans tous les cas, il a été obtenu par essai et erreur.

8.3.2 Mesure de l'exactitude de la prédiction

La mesure la plus commune pour des approches probabilistes est la vraisemblance des données ou des approximations telles que le BIC (Bayesian Information Criterion). Néanmoins pour notre problème particulier, cette métrique a l'inconvénient de ne pas avoir une interprétation géométrique. Intuitivement, nous voudrions connaître la distance entre l'état prédit et le réel. Donc, nous avons préféré de mesurer la performance de notre approche en termes de l'erreur moyenne, calculée comme l'expectation de la distance entre la prédiction pour un horizon temporel donné H et l'observation effective O_{t+H}.

$$\langle E \rangle = \sum_{i \in \mathcal{S}} P([S_{t+H} = i] \mid O_{1:t}) \| O_{t+H} - \mu_i \|^{1/2} \tag{47}$$

Pour un seul pas de temps. Cela peut être généralisé à un ensemble de données complet, contenant K séquences d'observations.

$$\langle E \rangle = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{T^k - H} \sum_{t=1}^{T^k - H} \sum_{i \in \mathcal{S}} P([S_{t+H} = i] \mid O^k_{1:t}) \| O^k_{t+H} - \mu_i \|^{1/2} \tag{48}$$

Une autre différence par rapport a l'approche standard en apprentissage automatique, est que, à la place d'utiliser deux ensembles de données pour nos expériences - l'un pour l'apprentissage, l'autre pour la prédiction - nous avons utilisé un seul ensemble de données. Cela est possible, car l'apprentissage est réalisé juste après la prédiction, et de ce fait, chaque séquence d'observations est, effectivement, "inconnue" quand la prédiction a lieu.

Le reste de ce chapitre est dédié à la présentation des résultats quantitatifs obtenus avec les différents ensembles de donées.

8.3.3 Données réelles du hall d'entrée

8.3.4 Données synthétiques du hall d'entrée

8.3.5 Données du parking de Leeds

8.3.6 Données du parking de l'INRIA

Chapitre 9: Conclusions et perspectives

Dans ce chapitre, nous récapitulons la situation générale des approches de prédiction basées sur les mouvements typiques et rappelons nos contributions. Après, nous étudions les perspectives de notre travail.

9.2 Perspectives et extensions possibles

Nous voyons deux directions principales dans lesquelles les travaux présentés dans cette thèse peuvent se développer. La première consiste à explorer des extensions vers des niveaux plus hauts d'abstraction, spécifiquement en prenant en compte les interactions qui peuvent arriver entre des objets mobiles. La deuxième consiste à améliorer le bas niveau pour augmenter la robustesse de notre approche face aux limitations des systèmes de suivi et des capteurs actuels.

9.2.1 Extensions de haut niveau

Notre approche ne prend pas en compte la possibilité qu'un objet modifie sa trajectoire en réponse aux mouvements d'un autre objet. Un premier pas dans cette direction pourrait être de considérer les objets semi-statiques que l'on trouve dans un environnement. Ces objets se caractérisent par le fait qu'ils ne peuvent pas se déplacer librement, mais peuvent adopter un de deux états possibles (blockage/non blockage). C'est le cas, par exemple, des portes, qui peuvent être soit ouvertes, soit fermées, ou des places de parking, qui peuvent être soit libres, soit occupées. Nous illustrons comment ces objets peuvent être pris en compte, en introduisant la notion d'état occupé dans notre modèle.

Un défi beaucoup plus ambitieux est celui de prendre en compte les interactions entre des objets complètement dynamiques. Dans la base, cela requiert de modéliser l'état conjoint de tous les objets qui occupent l'environnement, ce qui est manifestement non faisable. Nous esquissons deux directions possibles: soit en ayant recours à des techniques qui décomposent l'état conjoint [Brand et al., 1997, Gong and Xiang, 2003], soit en intégrant notre approche avec des techniques d'analyse de scènes [Oliver et al., 2000].

9.2.3 Extensions de bas niveau

Comme la plupart des approches basées sur des comportements typiques, la nôtre repose sur l'hypothèse qu'on dispose d'un système de suivi pratiquement sans failles. Seulement, comme

nous l'avons montré dans le chapitre 7, cette hypothèse n'est pas vérifiée dans la pratique.

Nous voyons deux stratégies pour résoudre ce problème, la première est de développer des systèmes de diagnostic qui nous permettent de filtrer les trajectoires "défectueuses" [Hu et al., 2006]; l'autre est d'essayer d'améliorer le suivi en utilisant des comportements appris [Liao et al., 2003, Bennewitz et al., 2005].

Chapter 1

Introduction

Life is a series of collisions with the future

JOSE ORTEGA Y GASSET

1.1 Motivation

Motion planning for dynamic environments is a very active research domain in robotics. Owing to the fact that the problem is NP-Hard [Reif and Sharir, 1985], most research efforts have been directed towards coping with planning complexity.

There is, however, another critical aspect of the problem which is often overlooked: motion planning algorithms need to know in advance how the objects that populate the environment will move. In practice, this knowledge is seldom available, making it necessary to resort to prediction: gathering information about moving objects through a sensor device (*eg* radar, visual tracker, laser scanner) and then feeding this information into a mathematical model of the object's motion in order to obtain an estimation of its future state (*eg* position, velocity).

Until recently, most motion prediction techniques have been based on kinematic or dynamic models that describe how the state of an object evolves over time when it is subject to a given control (*eg* acceleration) (cf. [Zhu, 1990]). These approaches proceed by estimating the state, using techniques such as the Kalman Filter [Kalman, 1960], and then applying the estimate to its motion equations in order to get state predictions.

Although these techniques are able to produce very good short-term predictions, their performance degrades quickly as they try to see further away in the future. This is especially true for humans, vehicles, robots, animals and the like, which are able to modify their trajectory according to factors (*eg* perception, internal state, intentions, etc.) which are not described by their kinematic or dynamic properties.

To address this issue, a different family of approaches has emerged in the last decade. It is based on the idea that, for a given environment, moving objects tend to follow typical motion patterns that depend on the objects' nature and the structure of the environment. These approaches operate in two stages:

1. *Learning stage:* observe the moving objects in the workspace in order to identify and build representations of typical motion patterns.

2. *Prediction stage:* use the learned typical motion patterns to predict the future motion of a given object.

Thus, learning consists in observing a given environment in order to construct a representation of every possible motion pattern. But, how long should we observe the environment in order to construct such a "library" of motion patterns? Given the enormous number of possible behaviors the humans may exhibit in all but the simplest environments, there is not a simple answer. This raises an important problem of most current learning techniques [Hu et al., 2004a, Bennewitz et al., 2005, Wang et al., 2006]: they use a "learn then predict" approach, meaning that the system goes through a learning stage where it is presented with an example data set from which it builds its pattern models. Then, the models are "frozen" and the system goes into the prediction stage.

The problem with this approach is that it makes the implicit assumption that there is enough information in the example data set to build a representation of every possible motion pattern, which, as we have shown, is a difficult condition to guarantee. This thesis proposes a solution to this problem: a *learn and predict* approach, where learning and prediction take place in a continuous and parallel fashion, making it possible to refine knowledge on the basis of the same observations that are used to predict motion as it happens.

1.2 Problem description

It will be assumed hereafter that the input of a learning based algorithm consists of a series of discrete observations (*ie* sensor readings) describing the motion of objects in a given environment; most of the time, we will consider observations to consist of position information in the form of coordinates in the plan, nevertheless, sometimes they also include other information like velocity, size, orientation, confidence, etc.

In addition to this information, it is assumed that observations are arranged in sequences $O_{1:T} = \{O_1, \cdots, O_T\}$ such that every sequence describes the trajectory of a single object since it observed for the first time until it leaves the environment or stops moving. For example, in the case of a visual tracker, observation sequences will correspond to complete tracks, from creation until deletion.

A complete learning-based approach, includes the following components:

A *motion pattern model* describing, how the object's state evolves as time passes, considering that it is engaged in a given motion pattern.

A *learning algorithm* specifying how the model's parameters should be estimated from collected data. For a "learn and predict" approach, this should ideally comprise the ability to create or delete motion pattern models as learning progresses.

A *prediction algorithm* prescribing the use of learned models and observations to predict future motion. More precisely it should answer the following question: given our current

knowledge and the whole history of observations for an object up to the present time t, what will be its state at time $t+H$?

Since uncertainty is inherent to prediction, it seems sensible to use a probabilistic framework and to model motion patterns as stochastic processes. Thus, we have based this thesis on Hidden Markov Models [Rabiner, 1990], a probabilistic approach which is very popular in the literature of "learn then predict" approaches [*eg* Walter et al., 1999, Makris and Ellis, 2002, Bennewitz et al., 2005] due, at least in part, to the existence of efficient inference and learning algorithms. Nevertheless, the application of Hidden Markov Models (HMM) to a "learn and predict" approach, poses additional problems which, as we will see in the next section, are not solved by classical algorithms.

1.2.1 Modeling motion with Hidden Markov Models

A Hidden Markov Model may be viewed as a graph (fig. 1.1), where nodes represent discrete states (*eg* places in the environment) and edges represent transitions. The topology of this graph (*ie* its edges) and its cardinality (*ie* number of states) are often called the *model's structure*. The model assumes that the state of an object is not accessible in a direct fashion, instead, there is an *observation probability*, attached to every state, describing the probability of obtaining a given observation, supposing that the object is in that state. Transitions are not deterministic either, but occur according to a *transition probability*, attached to every edge on the graph. Transition and observation probabilities are often represented as multinomial and Gaussian probabilities, respectively, the set of all the multinomial and Gaussian parameters are known together as the *model's parameters*.

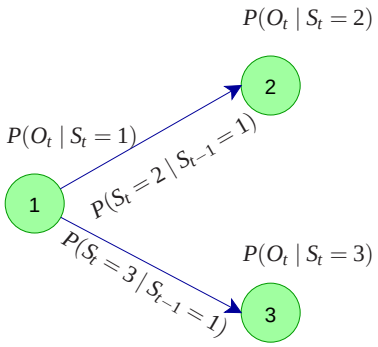


Figure 1.1: A basic three-state HMM, with transition probabilities attached to edges and observation probabilities attached to nodes.

HMM based motion prediction approaches in the literature model motion patterns as *typical trajectories*, which, in HMMs are represented as chains (*ie* order-two graphs) with directed edges going only in one direction (fig. 1.2). Every such graph constitutes an individual HMM

[*eg* Walter et al., 1999]; or, alternatively, they may be represented as a single HMM conformed by many non-connected chains [*eg* Bennewitz et al., 2005].

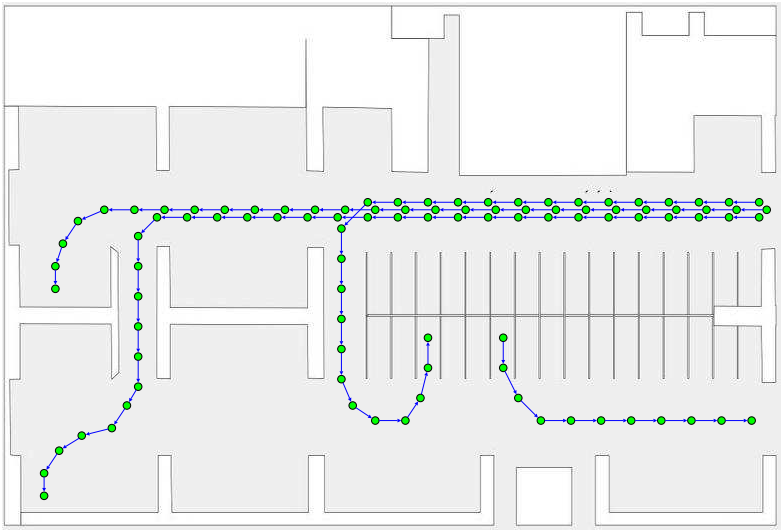


Figure 1.2: An HMM structure for a parking environment, where motion patterns are represented as non connected chains (order 2 subgraphs) (only a few motion patterns are displayed).

1.2.2 Challenges

When using HMMs as motion models there are actually two learning tasks to be accomplished: a) learn the graph structure, and b) estimate the parameters of the different probability distributions. In most cases, they are roughly equivalent to *identifying* and *representing* motion patterns, respectively.

The standard learning technique for HMMs is the Baum-Welch algorithm Baum et al. [1970] – a specialization of the well known Expectation Maximization algorithm [Dempster et al., 1977] – unfortunately, it is a parameter-only learning algorithm which assumes that the model's structure is known *a priori*. Moreover, in its classical form, the algorithm is not applicable to a "learn and predict" approach, because it is designed for off-line use, processing data in batch mode.

Although structure-learning and incremental extensions of the Baum-Welch algorithm exist [Friedman, 1997, Singer and Warmuth, 1996], an algorithm that is able to perform both of them is still needed in order to enable an HMM based "learn and predict" approach. Moreover, the algorithm should be able to work in *real time* which, in this context, means delivering predictions in the time elapsed between two sensor readings.

Prediction in HMMs consists in applying Bayesian inference to update a probabilistic belief (*ie* the belief state) of the object's state with newly gathered observations, and then to project it into the future. This may be done very efficiently on an observation by observation basis,

however, depending on the HMM's structure, inference may not be fast enough to enable real-time use, and it is necessary to apply approximate methods [*eg* Walter et al., 1999].

Another problem comes from the fact that, in real life, persons often change their mind while moving, leading to trajectories which are highly atypical and, at the same time, are composed by well known sub-behaviors. The problem with most current approaches is that they are not able to deal with this "mixed patterns" because typical trajectories are supposed to be completely independent. This leads to bad generalization, since a new motion pattern should be learned for every such case, which is problematic because will not represent actual behaviors, but an atypical motion which is unlikely to observe again.

1.3 Contributions

The main contribution of this thesis is a "learn and predict" technique based on a proposed extension of Hidden Markov Models, called Growing Hidden Markov Models (GHMM). They differ from conventional HMMs in the fact that the model structure and parameters are not fixed, but evolve continuously, as more observations are available. This is possible thanks to the integration of a topology learning network [Jockusch and Ritter, 1999] – which is used to learn the structure – and an incremental parameter learning algorithm inspired on the work of Neal and Hinton [1998] into a fully unsupervised learning technique.

Even if they have been formulated in the context of motion prediction, GHMMs are intended to be applicable to all the cases in which standard HMMs are used as a discrete approximation to model a process having a continuous state space.

In order to apply GHMMs to the motion prediction problem, we have taken a different approach from other HMM based techniques, which – as explained above – are based in the concept of typical trajectories. Instead, we have chosen to model motion in terms of the object's intended destination (*ie* its goal), which, in our approach, becomes part of an extended state vector, along with the other state variables such as pose and velocity.

By using together GHMMs and the extended state, we have developed a novel motion prediction technique, which has the following differences with respect to other HMM-based approaches:

- The model's parameters and structure are estimated using an incremental algorithm, which is a necessary condition for a true "learn and predict" approach.
- The learning algorithm is fully unsupervised, and it is defined in terms of intuitive parameters, which do not require any prior knowledge about the environment or the number of possible motion patterns.
- The learned HMM structure does not consists of unconnected chains representing 'typical trajectories' corresponding to geometrically similar observation sequences. Instead, motion is modeled in terms of goals. This not only enables richer motion pattern representations, but also has the advantage of being an explicit – although rough – model of the object's intentions.

- In spite of being richer than in other approaches, the learned HMM structure is still simple enough to enable the use of exact inference in real time, even for large environments and high-dimensional spaces: in most cases, the number of edges grows linearly with the number of discrete states in the model.

1.4 Overview of the rest of this thesis

Part One: Background

Chapter 2. This chapter introduces the basic mathematical notions of probabilities and inference which are fundamental for the rest of this thesis.

Part Two: State of the art

Chapter 3. This chapter is a review of the literature of learning-based motion models and their application to prediction. It proposes a classification of motion models, analyzes existing techniques, and discusses outstanding issues on intentional motion prediction, making emphasis in problems related to discrete space-state representations such as Hidden Markov Models.

Chapter 4. Since most of our work is based on Hidden Markov Models, we have consecrated one chapter to this probabilistic tool. In it, we discuss common inference tasks like filtering, smoothing and prediction; and present an overview of existing parameter and structure learning algorithms.

Part Three: Proposed approach

Chapter 5. This chapter introduces Growing Hidden Markov Models, our proposed HMM extension for continuous parameter and structure learning. It introduces the concept of a topological map and provides an overview of Vector Quantization and Topology Representing Networks, before presenting the Instantaneous Topological Map (ITM). It then formalizes Growing Hidden Markov Models, and explains how the ITM algorithm is integrated into an incremental structure and parameter learning algorithm.

Chapter 6. This chapter explains the application of GHMMs to predict the motion of vehicles and pedestrians. It explains the reasons and advantages of using an extended state vector and describes how it is integrated into the model. Finally, it compares our proposed approach against other HMM-based techniques in the literature.

Part Four: Experiments and Applications

Chapter 7. This chapter presents the different experimental platforms used in this thesis as well as the respective data sets.

Chapter 8 This chapter presents the qualitative and quantitative results that we have obtained from the application of our GHMM based motion prediction technique to the data sets described in the precedent chapter.

Part Five: Conclusions

Chapter 9. Finally, this chapter summarizes this thesis and analyzes our contributions. Then, it presents our conclusions and reviews some of the possibilities for future work.

Part I

Background

Chapter 2

Probabilistic Models

- How do you know?
- All those movies had happy endings.
- All?
- Most.
- That cuts down the probability - he told her, smug.

THOMAS PYNCHON
The crying of lot 49

2.1 Overview

Probabilities and Bayesian models – in particular the Bayes filter – will play a rather predominant role in what is to follow, so that it seems advisable to begin with a summary introduction to the subject. This chapter is strongly based on Lebeltel [1999], the interested reader is also invited to consult the works of Stone et al. [1999] and Thrun et al. [2005] for a more thorough explanation of the concepts presented here.

We place ourselves in the framework of the theory of plausible reasoning developed by Jaynes [1995] "probability as logic"; an extension of classical logic which introduces probabilities as a way to perform inference. One of the key aspects of this theory is its interpretation of probabilities as describing the degree of certainty that it is possible to have about a particular phenomenon given the available knowledge. This *subjective* approach is a significant departure from the classic *frequentist* point of view, which sees probabilities as an inherent property of the phenomenon.

The basis of Jaynes work is a theorem proposed by Cox [1946], who first stated a number of desiderata defining the concept of plausibility for a reasonable agent. From there, he showed that the only way to manipulate this notion while staying true to the original desiderata is by using the mathematical concept of probabilities. This result is crucial for a "scientifically respectable theory of inference"[1]: although estimating probabilities is subjective because it depends on the

[1] [Jaynes, 1995, p.39]

agent's knowledge, given the same knowledge, two different agents should estimate the same probabilities if they are rational.

2.2 From logic to probabilities

2.2.1 Logic Propositions

We will work with classic logic propositions which may be either true or false. These propositions will be denoted using capital calligraphic characters. Propositions may be manipulated using Boolean operators.

- *Logical AND*. We will denote the logical product, or conjunction, by listing propositions together, separated by spaces, for example $\mathcal{A}\,\mathcal{B}$ means "both $\mathcal{A}$ *and* $\mathcal{B}$ are true". Sometimes, for the sake of clarity, we will use a comma to indicate conjunction (*eg* "$\mathcal{A}\,,\mathcal{B}$"). Finally, as a notational convenience, we will use colon-separated sub indices to denote the conjunction of variables with consecutive indexes (*eg* $\mathcal{A}_{1:t} = \mathcal{A}_1, \cdots, \mathcal{A}_t$).

- *Logical OR*. Logical disjunction will always be denoted by using a plus sign, hence $\mathcal{A} + \mathcal{B}$ means "at least one of $\mathcal{A}$ *or* $\mathcal{B}$ is true".

- *Negation*. Negation will be denoted by the symbol $\neg$, therefore $\neg\mathcal{A}$ means "$\mathcal{A}$ is not true".

2.2.2 Probability of a proposition

Sometimes, we do not know if a particular proposition $\mathcal{A}$ is true or not, but we still have reasons (*eg* prior knowledge or evidence) that make us believe that one of those values is more likely than the other. We express this using the notation $P(\mathcal{A} \mid \pi)$ which is read "the conditional probability of $\mathcal{A}$ *given* our former knowledge π". Since – under the subjective interpretation – probabilities are always estimated on the basis of former knowledge, it does not make sense to simply write $P(\mathcal{A})$ or "probability of $\mathcal{A}$". That said, we will systematically omit the specification of former knowledge π, but it should be noted that this is only a notational shortcut.

Quantitative rules for propositions

Quantitative rules are mathematical identities used to manipulate proposition probabilities in order to perform inference, in a similar fashion to what we do with Boolean operators.

The product rule The product rule relates the probability of the logical product $\mathcal{A}\ \mathcal{B}$ to the individual probabilities of $\mathcal{A}$ and $\mathcal{B}$:

$$P(\mathcal{A}\ \mathcal{B}) = P(\mathcal{A})P(\mathcal{B} \mid \mathcal{A}) = P(\mathcal{B})P(\mathcal{A} \mid \mathcal{B}) \tag{2.1}$$

The normalization rule The normalization rule expresses the relationship between the probability of a proposition and that of its negation.

$$P(\mathcal{A}) + P(\neg\mathcal{A}) = 1 \tag{2.2}$$

The sum rule Just as in classical logic it is possible to construct every possible logical expression using only negation and conjunction, two identities – the product and the normalization rules – are used to derive all possible probabilistic computations. One of the rules which is possible to obtain from (2.1) and (2.2) is the sum rule:

$$P(\mathcal{A} + \mathcal{B} \mid \mathcal{C}) = P(\mathcal{A} \mid \mathcal{C}) + P(\mathcal{B} \mid \mathcal{C}) - P(\mathcal{A}\ \mathcal{B} \mid \mathcal{C}) \tag{2.3}$$

2.2.3 Variables

Until now, we have only talked in terms of logical propositions but, most often, we want to reason in terms of variables (also called random variables) which represent the relevant features of a given problem. A discrete variable V has an associated domain $D_V = \{v_1, \cdots, v_K\}$, which is the set of K values which this variable may take. The variable may only have a single value at the same time, and the value should be in D_V.

Let us suppose that we want to model the result of throwing a die, representing the output by a discrete variable V having a domain $D_V = \{1, \cdots, 6\}$. We may associate a logic proposition with every possible output:

$$\begin{aligned} \mathcal{V}_1 &\equiv [V = 1] \\ &\vdots \\ \mathcal{V}_6 &\equiv [V = 6] \end{aligned}$$

These propositions are both *exhaustive* and *mutually exclusive*.

$$\begin{aligned} \mathcal{V}_1 + \cdots + \mathcal{V}_6 &= 1 \,\text{(exhaustive)} \\ \mathcal{V}_i \mathcal{V}_j &= 0\ \forall i, j \mid i \neq j \,\text{(mutually exclusive)} \end{aligned}$$

We may now use the obtained propositions to define probabilities, for example, the probability of obtaining a 6 may be written $P(\mathcal{V}_6)$, alternatively, we may explicitly enclose the proposition with square brackets $P([V = 6])$.

If we have two variables V^1 and V^2 whose respective domains are $D_{V^1} = \{v_1^1, \cdots, v_N^1\}$ and $D_{V^2} = \{v_1^2, \cdots, v_M^2\}$, we may describe $N \times M$ mutually exclusive elementary propositions:

$$
\begin{gathered}
\mathcal{V}_1^1\mathcal{V}_1^2, \mathcal{V}_1^1\mathcal{V}_2^2, \cdots, \mathcal{V}_1^1\mathcal{V}_N^2 \\
\mathcal{V}_2^1\mathcal{V}_1^2, \mathcal{V}_2^1\mathcal{V}_2^2, \cdots, \mathcal{V}_2^1\mathcal{V}_N^2 \\
\vdots \\
\mathcal{V}_M^1\mathcal{V}_1^2, \mathcal{V}_M^1\mathcal{V}_2^2, \cdots, \mathcal{V}_M^1\mathcal{V}_N^2
\end{gathered}
$$

Each of these $N \times M$ propositions may be interpreted as follows:

$$
\mathcal{V}_i^1\mathcal{V}_j^2 \equiv [V^1 = v_i^1][V^2 = v_j^2]
$$

"The value of V^1 is v_i^1 *and* the value of V^2 is v_j^2"

Since these propositions are both exhaustive and mutually exclusive, the conjunction $V^1\,V^2$ is itself a new random variable.

Quantitative rules for discrete variables

Whereas there are some differences between variables with discrete and real domains, we will treat them individually. For discrete variables, the product rule is:

$$
\begin{aligned}
&\forall\, i \in D_A,\ j \in D_B \\
&P([A=i]\ [B=j]) = P([A=i])P([B=j] \mid [A=i]) \\
&\qquad\qquad = P([B=j])P([A=i] \mid [B=j])
\end{aligned}
\tag{2.4}
$$

In this document, we will use the variable name alone (*ie* without square brackets) to denote the *entire* probability distribution. Hence the above expression may be noted with the following shortcut:

$$
P(A\,B) = P(A)P(B \mid A) = P(B)P(A \mid B) \tag{2.5}
$$

In general, the probability of the conjunction of two or more variables is called a Joint Probability Distribution (JPD). The factorization of the JPD into simpler probabilities is called a *decomposition*.

The normalization rule may at first look somewhat different from (2.2). Intuitively, both of them may be regarded as stating that the sum of the probabilities for all the possible cases is one:

$$
\sum_{i \in D_A} P([A=i]) = 1 \tag{2.6}
$$

Again, this may be written:

$$
\sum_A P(A) = 1 \tag{2.7}
$$

Quantitative rules for continuous variables

Of course, many problems are defined in terms of variables having a continuous domain. In this case, probabilities are defined by assuming that there is a Probability Density Function (PDF) $g(A)$ which defines the probability of an interval:

$$P([a < A \leq b]) = \int_{A=a}^{b} g(A)dA \tag{2.8}$$

For continuous variables, the product rule is the same that in (2.5):

$$P(A\,B) = P(A)P(B \mid A) = P(B)P(A \mid B) \tag{2.9}$$

And the normalization rule becomes:

$$\int_{-\infty}^{\infty} g(A)dA = 1 \tag{2.10}$$

Other useful identities

A very useful identity involving variables is the *theorem of total probability*, also known as the *marginalization rule*:

$$\sum_{B} P(A\,B) = \sum_{B} P(B)P(A \mid B) = P(A) \qquad \text{(discrete case)} \tag{2.11}$$

$$\int_{-\infty}^{\infty} P(A\,B)dB = \int_{-\infty}^{\infty} P(B)P(A \mid B)dB = P(A) \qquad \text{(continuous case)} \tag{2.12}$$

Equally important is the so called *Bayes rule*, which establish the relationship between the "forward probability" $P(A \mid B)$ and its "inverse" $P(B \mid A)$, this is particularly useful in inverse problems, where the parameters of a model need to be estimated from data:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} = \frac{P(B \mid A)P(A)}{\sum_A P(B \mid A)P(A)} \qquad \text{(discrete case)} \tag{2.13}$$

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} = \frac{P(B \mid A)P(A)}{\int_{-\infty}^{\infty} P(B \mid A)P(A)dA} \qquad \text{(continuous case)} \tag{2.14}$$

It is important to note that, since the denominator does not depend on A

$$P(A \mid B) \propto P(B \mid A)P(A) \tag{2.15}$$

and the proportionality coefficient may be deduced from the marginalization rule.

The Bayes rule may be seen as a mechanism to compute (*ie* infer) the probability of a variable A, whose value we ignore, at the light of a variable B, whose value is known. The known variable is often called the *evidence*, the term $P(A \mid B)$ is called the posterior probability or simply the *posterior* and the term $P(A)$ is called the prior probability or simply the *prior*. Finally, the term $P(B \mid A)$ is often called *data likelihood*[2].

[2]There is a lot more to say about Bayes rule. For example, the importance of priors and their use to describe

2.2.4 JPD Decomposition and conditional independence

The use of the quantitative rules is not limited to the case of two variables: since the product of two or more random variables is itself a random variable, it is possible to apply the rules to an arbitrary number of variables.

The dependencies between different variables are formally denoted by their joint probability, and the way it is decomposed into simpler probability distributions by applying the product rule. For example, for the case of three variables $\{V^1, V^2, V^3\}$ there are 13 possible decompositions of the JPD:

$$\begin{aligned}
P(V^1, V^2, V^3) &= P(V^1)P(V^2\ V^3 \mid\ V^1) \\
&= P(V^2)P(V^1\ V^3 \mid\ V^2) \\
&= P(V^3)P(V^1\ V^2 \mid V^3) \\
&= P(V^1\ V^2)P(V^3 \mid V^1\ V^2) \\
&= P(V^1\ V^3)P(V^2 \mid V^1\ V^3) \\
&= P(V^2\ V^3)P(V^1 \mid V^2\ V^3) \\
&= P(V^1)P(V^2 \mid V^1)P(V^3 \mid V^1\ V^2) \\
&= P(V^1)P(V^3 \mid V^1)P(V^2 \mid V^1\ V^3) \\
&= P(V^2)P(V^1 \mid V^2)P(V^3 \mid V^1\ V^2) \\
&= P(V^2)P(V^3 \mid V^2)P(V^1 \mid V^2\ V^3) \\
&= P(V^3)P(V^1 \mid V^3)P(V^2 \mid V^1\ V^3) \\
&= P(V^3)P(V^2 \mid V^3)P(V^1 \mid V^2\ V^3)
\end{aligned}$$

These 13 factorizations are equivalent from a mathematical point of view, however, by choosing a particular decomposition, we express our knowledge about the structure of variable dependence, as related to a particular problem or application.

After choosing a decomposition, further simplification is possible on the basis of *conditional independence* assumptions, which apply whenever we consider that the variable V^1 does not provide additional information about a variable V^2 if a third variable V^3 is known:

$$\begin{aligned}
P(V^1\ V^2\ V^3) &= P(V^1)P(V^2 \mid V^1)P(V^3 \mid V^1\ V^2) \\
&= P(V^1)P(V^2 \mid V^1)P(V^3 \mid V^1)
\end{aligned}$$

These assumptions are based on knowledge about the variables' semantics, they may also be introduced to simplify, even if they are not necessarily true. Anyway, conditional independence assumptions are critical for tractability since they permit to reduce the dimensionality and, therefore, the complexity of the distributions which conform the JPD.

---

former knowledge is a source of debate between frequentists and bayesians (*eg* subjectivists). However, such a discussion is beyond the scope of this thesis.

2.2.5 Inference

The rules we have described so far may be used to put together a model which describes some phenomenon or process; such a probabilistic model is specified by enumerating the variables – and their domains – which are considered to be relevant to the phenomenon being modeled, as well as the corresponding JPD decomposition.

The main application of probabilistic models is inference: finding out the values of unknown variables on the basis of known variables (*ie* evidence) through the application of the Bayes rule. More formally, inference consists in finding the probability distribution of a set of variables ε_r (*ie* researched variables) on the basis of another set of variables ε_k, whose value we know (*ie* evidence), eventually disregarding the values of a third group of variables ε_i (*ie* ignored variables). We define a probabilistic question as any expression having the form:

$$P(X^k \cdots X^l \mid [X^m = x^m] \cdots [X^n = x^n]) \tag{2.16}$$

Where $\varepsilon_r = \{X^k, \cdots, X^l\} \neq \varnothing$, $\varepsilon_k = \{X^m, \cdots, X^n\}$ and $\varepsilon_i = \{X^o, \cdots, X^p\}$ is the set of variables that do not belong to neither ε_r nor ε_k.

Knowing the JPD, it is possible to answer any probabilistic question of the form (2.16) by first applying the product rule (2.1):

$$P(X^k \cdots X^l \mid [X^m = x^m] \cdots [X^n = x^n]) = \frac{P(X^k \cdots X^l\, [X^m = x^m] \cdots [X^n = x^n])}{P([X^m = x^m] \cdots [X^n = x^n])} \tag{2.17}$$

And then expressing the numerator and the denominator in terms of the JPD by applying the marginalization rule (2.12):

$$\cdots = \frac{\sum_{X^o \cdots X^p} P(X^k \cdots X^l\, [X^m = x^m] \cdots [X^n = x^n]\, X^o \cdots X^p)}{\sum_{\substack{X^k \cdots X^l \\ X^o \cdots X^p}} P(X^k \cdots X^l\, [X^m = x^m] \cdots [X^n = x^n]\, X^o \cdots X^p)} \tag{2.18}$$

As Cooper [1990] has demonstrated, Bayesian inference is NP-hard; this explains the importance of conditional independence assumptions, which render the problem tractable by allowing further simplifications. Another way to cope with complexity is by using specially tailored inference algorithms which exploit specific features of a model; it is also possible to trade-off accuracy against complexity by using approximate inference algorithms. Some examples of these algorithms will be given in § 2.3.

2.2.6 Parametric forms

Up to this moment, we have not really explained how the simpler probability distributions that compose the JPD are defined; indeed, a complete probabilistic model description should specify how these probabilities are represented. Throughout this thesis, these probabilities will be chosen amongst a few elementary distributions, which are defined in terms of a number of parameters, and hence are also known as *parametric forms*.

Uniform Distribution

The uniform distribution (noted $\mathbf{U}_V$) represents the fact that all the values in the domain of a variable V are equiprobable. In general, it is only well defined for discrete variables:

$$P([V = v_i]) = \mathbf{U}_V(v_i) = \frac{1}{|D_V|}, \forall v_i \in D_V \tag{2.19}$$

The only parameter of this distribution is the cardinality of the variable's domain $|D_V|$.

Conditional Probability Table

Conditional probabilities on discrete variables may be represented as a Conditional Probability Table (CPT), which lists the probabilities that the variables on the left side of the bar may take depending on the evidence variables. for example, if a, b and c are all binary variables (*ie* $D_a, D_b, D_c \in \{0,1\}$), the probability of a given b and c is represented with a $2 \times 2 \times 2$ table t whose individual elements will be represented with comma-separated sub indices:

$$P([A = i] \mid [B = j]\ [C = k]) = T_{i,j,k}$$

In general a conditional probability distribution consisting of N variables $\{V^1, \cdots, V^N\}$ will correspond to a N dimensional CPT with $|D_{V^1}| \times \cdots \times |D_{V^N}|$ elements or parameters.

Gaussian distribution

For continuous variables will be often use the multivariate Gaussian (or Normal) probability distribution, which has the following form[3]:

$$P([V = v_i]) = \mathbf{G}(v_i;\ \mu,\ \Sigma) \tag{2.20}$$

$$= |2\pi\Sigma|^{-1/2} \exp\left[-\frac{1}{2}(V-\mu)^T \Sigma^{-1}(V-\mu)\right] \tag{2.21}$$

where μ is the mean vector, Σ is a $D \times D$ positive semidefinite matrix called *covariance matrix* and $v_i, \mu \in \mathbb{R}^D$. The mean and covariance matrix[4] constitute the Gaussian's parameters.

2.2.7 Learning

Besides the definition of the probabilistic model itself, it is necessary to assign values to the parameters (*eg* mean value and covariance) of every elementary probability distribution in the decomposition. Although this may be done manually, it is also possible to learn (*ie* estimate) the parameters' values from data.

[3] Strictly speaking, the expression presented here does not denote a probability since, as it may be seen in eq. (2.8), continuous probability distributions are only defined for intervals. However, it is common in the literature to refer to PDFs as probabilities [*eg* Thrun et al., 2005], which is what we will do in this document.

[4] In the scalar (unidimensional) case, the covariance is called simply variance.

For discrete variables, in its simplest form, learning consists in computing the frequencies at which different variable's values occur in data, this is done by counting. So, if we have only one variable V with domain D_V:

$$P([V = v_i]) = \frac{C([V = v_i])}{\sum_V C(V)} \tag{2.22}$$

where $C(\mathcal{A})$ stands for the number of cases in data where proposition $\mathcal{A}$ is true. This is known as a *maximum likelihood* estimate, since it maximizes data likelihood without taking into account any prior on the distribution of V. A problem with this approach is that, when a learning case does not appear in data, the corresponding probability becomes zero. This disregards the – very frequent – situation in which data is not exhaustive, *ie* it does not contain an example of every possible value of V. The problem may be solved by using Dirichlet priors, that is, pre assigning counts α_i to every value v_i in D_V:

$$P([V = v_i]) = \frac{C([V = v_i])) + \alpha_i}{\sum_{v_j \in D_V} C([V = v_j]) + \alpha_j} \tag{2.23}$$

the particular case in which all $\alpha_i = 1$, becomes equal to a uniform in the absence of data; it is known as Laplace's law of succession:

$$P([V = v_i]) = \frac{C([V = v_i]) + 1}{\sum_V C(V) + 1} \tag{2.24}$$

Laplace's law may be generalized to learn an arbitrary conditional probability:

$$\begin{aligned} &P([V^1 = v^1] \cdots [V^k = v^k] \mid [V^l = v^l] \cdots [V^m = v_m]) \\ &\quad = \frac{C([V^1 = v^1] \cdots [V^k = v^k]\,[V^l = v^l] \cdots [V^m = v_m]) + 1}{\sum\limits_{V^1,\cdots,V^k} C(V^1 \cdots V^k\,[V^l = v^l] \cdots [V^m = v_m]) + 1} \end{aligned} \tag{2.25}$$

In many real situations, data is not available for some of the variables in the model, which are then said to be *hidden*. The standard solution is the use of the Expectation-Maximization (EM) algorithm [Dempster et al., 1977], which starts with some initial (often random) estimate of the parameters and then uses inference to compute the expected number of counts – called Expected Sufficient Statistics (ESS) – which are then treated as though they were observed and used to re-estimate the parameters. This two steps are iterated until some convergence criterion is met.

Another – and much more challenging – learning problem is conditional structure learning, which consists in estimating from data the best JPD decomposition and conditional independence assumptions. The general approach is to evaluate different decompositions using a scoring function and choose the one which obtains the best score.

Since the literature in learning probabilistic models is huge, we will not further discuss general methods in this chapter[5]. Instead, in chapter 4 we will do a more thorough review of the learning algorithms that apply to the particular probabilistic model upon which we have based our work: *Hidden Markov Models*.

[5] See [Heckerman, 1995, Murphy, 2002] for good introductions to the subject.

2.3 The Bayes filter

We will conclude this chapter by introducing the Bayes filter, which is used to study dynamic systems and constitutes the basic probabilistic framework for chapters 4 and 5.

The objective of the Bayes filter is to find a probabilistic estimate of the current state of a dynamic system – which is assumed to be "hidden", *ie* not directly observable – given a sequence of observations gathered every time step until the present moment.

The main advantage of using a probabilistic framework such as the Bayes filter is that it allows to take into account the different sources of uncertainty that participate in the process, such as:

- The limited precision of the sensors used to obtain observations.
- The variability of observations due to unknown factors (observation noise).
- The incompleteness of the model.

2.3.1 Probabilistic Model

Variables

The Bayes filter works with two types of variables[6]:

- S_t, the state of the system at time t. The exact meaning of this variable depends on the particular application, in general, it may be seen as the set of system features which are relevant to the problem and have an influence in the future. Since this work deals mostly with object motion, we will frequently assume that the state is the object's pose, although the specific variables that constitute the state may vary with the context and include additional features such as velocity. This will be explicitly stated in the text.
- O_t, the observation gathered at time t. Observations provide indirect indications about the state of the system. In this thesis, we will assume that observations come from sensors which "observe" a given environment, such as laser scanners and video trackers.

The Bayes filter is an abstract model which do not makes any assumption about the nature (*ie* discrete or continuous) of the state and observation variables. Such assumptions are made by concrete specializations of the Bayes filters, such as the Kalman Filter (continuous state and observations) or Hidden Markov models (discrete state, discrete/continuous observations).

Decomposition

A Bayes filter defines a joint probability distribution on $O_{1:T}$ and $S_{1:T}$ on the basis of two conditional independence assumptions:

[6] Here, we are providing the definition of Stone et al. [1999], which is different from the one given by Thrun et al. [2005] in that the later includes a third set of variables describing actions or controls.

1. Individual observations O_t are independent of all other variables given the current state S_t:

$$P(O_t \mid O_{1:t-1}\, S_{1:t}) = P(O_t \mid S_t) \tag{2.26}$$

   In general $P(O_t \mid S_t)$ is called *observation probability* or *sensor model*. It models the relationship between states and sensor readings, taking into account such factors as accuracy and sensor noise.

2. The current state depends only on the previous one, former states do not provide any further information. This is also known as the order one *Markov hypothesis*:

$$P(S_t \mid S_{1:t-1}) = \begin{cases} P(S_1) & \textit{for } t = 1 \\ P(S_t \mid S_{t-1}) & \textit{otherwise} \end{cases} \tag{2.27}$$

   The probability $P(S_t \mid S_{t-1})$ is called the *transition probability* or *system model*. The probability $P(S_0)$ which describes the initial state in the absence of any observations is called the *state prior*.

These assumptions lead to the following decomposition of the Bayes filter:

$$P(S_{1:T}\, O_{1:T}) = P(S_1)P(O_1 \mid S_1)\prod_{t=1}^{T} P(S_t \mid S_{t-1})P(O_t \mid S_t) \tag{2.28}$$

2.3.2 Parametric forms

The Bayes filter does not define any parametric forms and, consequently, no parameter identification mechanism. This is left out to specializations.

2.3.3 Inference

One of the main uses of Bayes filters is to answer the probabilistic question $P(S_{t+H} \mid O_{1:t})$; what is the state probability distribution for time $t+H$, knowing all the observations up to time t? The most common case is filtering ($H = 0$) which estimates the current state. However, it is also frequent to perform prediction ($H > 0$) or smoothing ($H < 0$).

The Bayes filter has a very useful property that largely contributes to its interest: filtering may be efficiently computed by incorporating the last observation O_t into the last state estimate using the following formula:

$$P(S_t \mid O_{1:t}) = \frac{1}{Z}P(O_t \mid S_t)\sum_{S_{t-1}} [P(S_t \mid S_{t-1})P(S_{t-1} \mid O_{1:t-1})] \tag{2.29}$$

where, by convention, Z is a normalization constant which ensures that probabilities sum to one over all possible values for S_t.

If we define recursively $P(S_{t-1}) = P(S_{t-1} \mid O_{1:t-1})$, it is possible to describe a Bayes filter in terms of only three variables: S_{t-1}, S_t and O_t, leading to the following decomposition:

$$P(S_{t-1}\ S_t\ O_t) = P(S_{t-1})P(S_t \mid S_{t-1})P(O_t \mid S_t) \tag{2.30}$$

Where the state posterior of the previous time step is used as the prior for the current time and is often called *belief state* Thrun et al. [2005].

Under this formulation, the Bayes filter is described in terms of a local model, which describes the state's evolution during a single time step. For notational convenience, in the rest of this thesis we are only going to describe such local models, noting that they always describe a single time step of the global model (2.29).

2.3.4 Specializations of the Bayes filter

Different choices of variable domains and parametric forms for the Bayes filter, give place to different filtering techniques. For example, Kalman Filters [Kalman, 1960] make the following hypotheses:

- State and observation variables are both continuous.
- The three probabilities in the decomposition are Gaussians.
- Both the evolution of the system state and the dependence between the system and the observation variables are described by linear functions.

Other popular specializations of the Bayes filter are Hidden Markov Models – which will be reviewed in detail in chapter 4 – and particle filters [Arulampalam et al., 2002], which represent probabilities using discrete samples, and allow for fast approximated inference. Particle filters will not be further discussed in this thesis.

2.4 Discussion

In this chapter we introduced the basic mathematical tools needed for understanding the following chapters. We have briefly presented the use of probabilities as a modeling and inference tool which allows the explicit representation of uncertainties.

We discussed the basic rules used to manipulate probabilities – *ie* the product and normalization rules – as well as some very useful derived rules – *ie* the sum and marginalization rules. We explained the components of a probabilistic model: the variables, joint probability decomposition and the parametric forms. We presented the Bayes rule and the fundamental role it plays in inference. We described the use and principles behind learning algorithms, in particular the importance of counting as a means of estimating probabilities. We have also introduced the Bayes filter and explained the variables that constitute it: state and observations.

Part II

State of the Art

Chapter 3

Intentional Motion Prediction

> Here is how it works: first you decide to treat the object whose behavior is to be predicted as a rational agent; then you figure out what beliefs that agent ought to have, given its place in the world and its purpose. Then you figure out what desires it ought to have, on the same considerations, and finally you predict that this rational agent will act to further its goals in the light of its beliefs. A little practical reasoning from the chosen set of beliefs and desires will in most instances yield a decision about what the agent ought to do; that is what you predict the agent will do.
>
> DANIEL DENNETT
> *The Intentional Stance*

3.1 Overview

In the epigraph, Dennett [1987] describes how most humans would try to predict another humans' motion if asked to: if we accept that a moving object is rational and that he is able to move at will – we take the *intentional stance* towards him – then it follows that the best way to predict its motion is to try to emulate its rational process, *eg* "the car is slowing down because he wants to park in the free place at the left". But designing a computer program capable of performing such emulation in a general fashion is beyond our current capabilities.

Instead, most intentional motion prediction approaches adopt what Dennet calls the *physical stance*[1]: they try to explain – and predict – the object's behavior in terms of its physical properties and the laws of physics. This implies the use of kinematic and dynamic models describing the objects' motion as the evolution of its state (*eg* its position, speed and orientation) over time when it is subject to a control (*eg* acceleration).

[1] It is important to note that the physical and intentional stances stand in opposite ends of a continuum instead of describing mutually exclusive views.

However, this approach faces at least three problems: a) the current object's state is not directly known, instead, it is necessary to estimate it from observations gathered through sensors which have a limited precision and are subject to noise; b) kinematic and dynamic models are inevitably incomplete, because, in order to be general, they deliberately disregard a number of factors that influence motion (*eg* road conditions, wind); and c) although objects following intentional motion certainly obey physical laws, they are able to change their control dynamically (*eg* steering angle, braking), hence to predict motion, it is necessary to predict the control that the object will apply.

The first problem may be addressed by using continuous Bayes filters such as the Kalman Filter [Kalman, 1960] and its extensions [*eg* Rao, 1996, Julier and Uhlmann, 1997]. However, this does not solve the other two problems, and, when applied to pedestrian and vehicles, motion prediction approaches based on Kalman filters [*eg* Han and Veloso, 1997, Rosales and Sclaroff, 1998, R.Madhavan and Schlenoff, 2003] produce predictions which are sound only during a short time interval – also known as its *time horizon* – or which are only applicable to highly structured environments. The same happens to approaches based on regression analysis [*eg* Elnagar and Gupta, 1998, Yu and Su, 2003, Liu et al., 2004], which have the additional drawback of producing deterministic predictions, thus being unable to represent the uncertainty that is inherent to the prediction process.

More recently, another family of approaches has emerged, based on an idea which is closer to the intentional stance: they assume that, in a given environment, objects do not move at random but often engage in typical behaviours or motion patterns. Therefore, if these motion patterns are known, it is possible to use them as motion models in order to predict motion.

This solves the third problem, since now it is only necessary to identify the motion pattern that the object is following, instead of predicting the applied control; indeed, it is not longer necessary to model controls. In addition, most of these approaches address the second problem by means of a probabilistic representation, where the inherent incompleteness of the model is reflected as uncertainty.

We will hereafter refer to this last family of approaches as "pattern based motion models"[2]. This chapter is a review of the related literature, it is structured according to a sub-classification of these approaches in three categories, depending on how they represent motion patterns:

1. *Trajectory Prototypes*. As their name suggests, these approaches model motion patterns using a trajectory prototype (*ie* typical example) for every motion pattern.

2. *Discrete state-space Models*. This models are based on *Markov chains*, where time progresses as discrete steps and, at every time step, the objects' state evolves by going from one state to another according to a given transition probability.

3. *Other representations*. Here we will discuss those approaches that do not fit into the other categories.

For every category of approaches, we are particularly interested in studying how they answer three questions:

[2] As opposed to "kinematic" or "dynamic" motion models

1. *Representation*. How are motion patterns described mathematically?

2. *Learning*. How are the model's parameters estimated from data?

3. *Prediction*. When an object is moving, what is the process that transforms observations into predictions?[3]

Throughout this chapter, when needed, we will base our discussion on the imaginary office environment depicted in fig. 3.1, where some pedestrian trajectories have been collected and are used as input for the different techniques discussed here. In this example, trajectory data is assumed to consist of noisy measurements of a person's pose, evenly sampled in time.

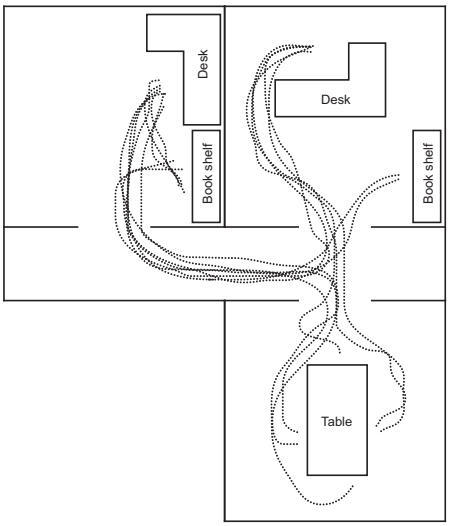


Figure 3.1: Raw trajectory data of people in an office environment.

After presenting existing approaches, we will conclude with a discussion of relevant global trends and issues in the field.

3.2 A note on semantics

Until now, we have indistinctly used the terms "behavior" and "motion pattern", this coincides with most of the reviewed works, which do not agree on a standard definition of these concepts. Henceforth, we will consider that a behavior describes what an object is doing by taking the intentional stance towards the object, while a pattern is the motion that may be observed when the object is involved in a particular behavior. We will illustrate this distinction with an example:

[3]We have decided to include in this review several approaches that do not specifically address the prediction question, but apply learning algorithms to build motion models, which, at least in theory, could be used to predict motion.

Let us imagine two persons in a hall, the first one is reading a bulletin board, while the second one is waiting for someone. From the persons' perspective these are two completely different behaviors, however, they produce very similar motion patterns: the persons do not move during a long time.

This distinction is important. At first sight it seems that, since the patterns are similar, they may be described by the same motion model. But this will lead us to dismiss a crucial difference: the "reading behavior" only happens in the bulletin board's neighborhood, while the "waiting behavior" may occur anywhere in the hall. As a consequence, our model will fail to represent the fact that it is much more likely to observe a motionless person near the bulletin board than elsewhere.

This is related to the notion of state. Most of the techniques that we have reviewed assume that the state variables represent only physical properties of an object such as its pose or velocity. Nonetheless, other representations are possible, as in [Oliver et al., 2000], where states have higher level semantics and objects may be in a "waiting" or a "fleeing" state and, therefore, are closer to modeling behavior.

3.3 Trajectory Prototypes

Approaches in this category work by grouping similar trajectories in clusters (fig. 3.2) – which correspond to typical motion patterns – then, for every cluster, a single trajectory prototype is obtained and used to represent the entire cluster and, in consequence, the corresponding motion pattern.

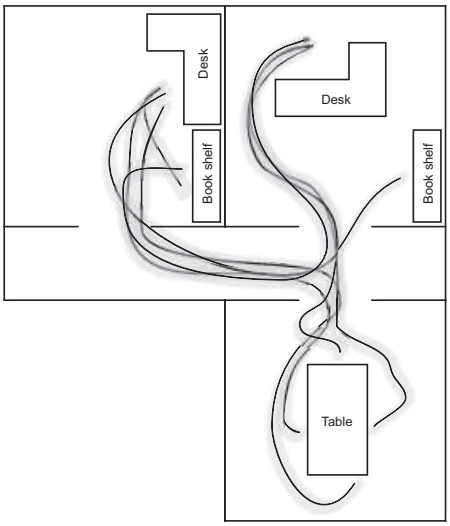


Figure 3.2: Trajectory prototypes for the office environment.

3.3.1 Representation

Typical trajectories are often represented as a sequence of points in the continuous state-space. Most approaches do not model the time variable explicitly and assume that the points in the sequence are regularly spaced in time.

Sometimes, a measure of the "width" of the cluster is also included in the representation (fig. 3.3). For example, Makris and Ellis [2001] construct a "path model" by including a deterministic measure of the width for every point in the trajectory, a similar measure has been used by Junejo et al. [2004]. Vasquez and Fraichard [2004] propose a probabilistic model in which the width of the trajectory is represented as the variance of the distance between the trajectories that belong to the same cluster. Another probabilistic model of width has been proposed by Bennewitz et al. [2002], they model every point of the trajectory prototype as a Gaussian and assume that all such Gaussians have the same covariance matrix.

3.3.2 Learning

Trajectory prototypes are obtained using classical clustering algorithms [see Kaufman and Rousseeuw, 1989, Jain et al., 1999]. The problems to be solved by a learning algorithm are basically three: a) determining the number of clusters to be found; b) finding clusters; and c) building trajectory prototypes from clusters, including, if necessary, the width representation.

There are many different classification schemes of clustering (*eg* hard vs. soft, deterministic vs. stochastic, see Jain et al. [1999]) here, we will distinguish between *model based clustering algorithms*, which work by deriving a set of prototype vectors; and *pairwise clustering algorithms*, which partition data into disjoint sets or clusters by performing pairwise comparisons between data elements. The interest of this classification is that algorithms of different categories provide different solutions to the three problems that we have mentioned above.

Model based algorithms

Model based algorithms have the advantage that trajectory prototypes are found as a part of the clustering process. They also have interesting theoretical properties since they are built by optimizing a global measure (*eg* data likelihood).

However, they need to know *a priori* the number of clusters – thus, of motion patterns – to be found, meaning that this number should be estimated somehow. Some approaches just ignore this problem, for example, Hu et al. [2004b] assume that the number of clusters is known and use a custom self organizing network to cluster trajectories. Others propose greedy search techniques, like, Bennewitz et al. [2002] which estimate the number of models by using the EM algorithm [Dempster et al., 1977] to cluster data using an initial guess of the number of clusters, and then adding or deleting clusters in order to maximize data likelihood, rerunning EM after every addition.

An issue with model based clustering is that most algorithms assume that their input consists of vectors of equal length, which is problematic because trajectory lengths may vary greatly. The most frequent solution [*eg* Hu et al., 2004b] is to resample trajectories in order to normalize their length. Unfortunately, this interferes with the assumption that the points in trajectory prototypes

are even spaced in time, and some temporal information is lost in the process. An alternative has been proposed in [Bennewitz et al., 2002], it consists in normalizing all the trajectories in the data set to the length of the longest trajectory by padding the end of shorter trajectories with copies of its last position; this may be interpreted as an object which stops moving, but is still detected for a while.

Pairwise Clustering

Pairwise clustering algorithms are based on the use of a dissimilarity measure (*eg* Euclidean distance) which is used to compare two data elements (*ie* trajectories). The value of the measure is high if the elements are very different and zero if they are equal. Data elements are processed in pairs and the dissimilarity measure is used to decide if elements belong to the same cluster. The clustering process produces groups of data elements, but no cluster representation. This is equivalent to adding a label to every element, indicating the group to whom it belongs. When compared with model based clustering algorithms, these approaches have the advantage that they are able to determine the number of clusters. On the other hand, since they are based in a local pairwise measure, it is not possible to guarantee that they satisfy a global optimality criterion.

In order to use pairwise clustering to learn motion patterns, it is necessary to choose a clustering algorithm, a dissimilarity measure and to device a mechanism to compute a cluster representation.

A popular pairwise clustering algorithm is Hierarchical Clustering [King, 1967], Makris and Ellis [2001] used it in conjunction with a maximum inter-point distance; while Buzan et al. [2004] used longest common subsequence (LCSS) as similarity measure; and Vasquez and Fraichard [2004], used a continuous version of Euclidean distance.

The use of graph cut algorithms [Boykov and Kolmogorov, 2004] has also been explored by Junejo et al. [2004], using the Hausdorff distance as dissimilarity measure. A custom pairwise algorithm has been proposed by Wang et al. [2006] based on a distance criterion augmented with a measure of confidence for observations, which permits to represent sensor noise and limited precision.

Once that clustering has been performed, the average trajectory for each cluster is computed and used to represent the cluster. Then, the cluster width may be computed either probabilistically – by computing a cluster variance [Vasquez and Fraichard, 2004] – or deterministically, by defining a cluster "envelope" [*eg* Junejo et al., 2004] which is obtained from the trajectories in the cluster that are farthest away from the average (fig. 3.3).

3.3.3 Prediction

Prediction using trajectory prototypes consists basically in finding the cluster which best corresponds to a partially observed trajectory – often using the same dissimilarity measure that has been used to cluster – and using that cluster's representation as a prediction of future motion. A variant has been proposed by Vasquez and Fraichard [2004] which returns a probability distribution on all the clusters according to their similarity.

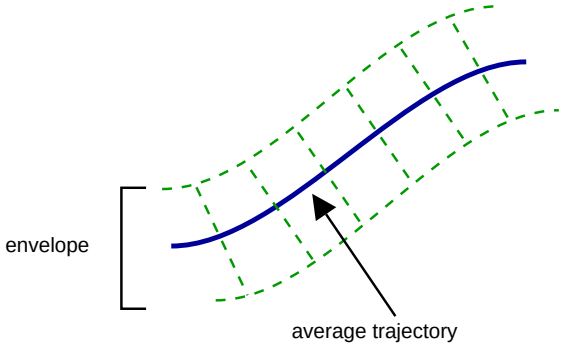


Figure 3.3: Example of average trajectory and deterministic envelope.

While these approaches have the advantage of being very coherent in the long-term – which, in theory, allows for very long time-horizons – they suffer from two important drawbacks: only trajectories that have been observed may be predicted; and, most important, they have a strictly deterministic representation of time, the predicted state for an object at time t corresponds to the t-th element of the trajectory prototype, which is too restrictive, given that velocities often vary between objects even if they are engaged in the same behavior. The problem becomes even more severe when trajectories are resampled to normalize their length. As a consequence, uncertainties may be handled only at the trajectory level, in other words, if the motion pattern that the object is following is known, these algorithms will provide a deterministic prediction of the state.

A solution to the last problem is to transform trajectory prototypes into state-space models, it will be discussed in §3.4.

3.4 Discrete state-space models

As their name indicates, these approaches are based on the use of a discrete model as an approximate analysis tool for continuous motion. The basis of discrete state-space approaches are Markov chains, which represent time as a discrete variable and model motion in terms of a finite number of discrete states. Transitions between states depend on a transition probability, which follows the Markov assumption, namely that, knowing the present state, the future and past states are independent. Markov chains may be represented by a directed graph, whose edges are labeled with the probability of going from one state to another in a single time step (fig. 3.4).

Markov chains, however, fail to model the uncertainty related to sensors, and only a few motion prediction techniques use them (*eg* Tadokoro et al. [1995], J. Rittscher and Stein [2003]). Observation uncertainty is specifically addressed by using discrete Bayes filters (cf. 2.3), which augment Markov chains by assuming that the state is not directly observable and should be inferred from sensor readings or *observations* to which they are related by an observation probability. One of the most popular discrete Bayes filters are Hidden Markov Models (HMM), which we have briefly described in the introduction and will be discussed in detail in chapter 4.

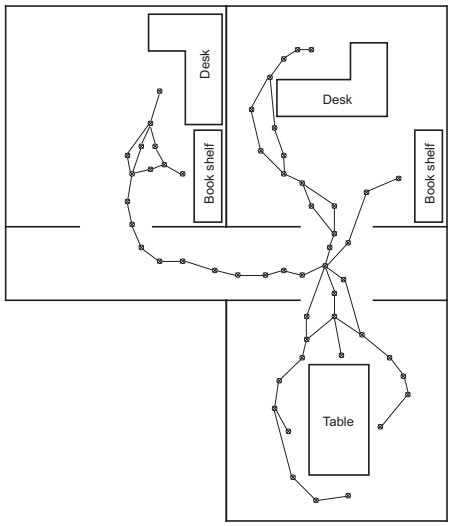


Figure 3.4: Discrete state-space model for the office environment (transition directions are not depicted).

Most of the techniques presented in this section are based on the HMM framework. However, they are also applicable to other discrete state-space models unless noted otherwise.

3.4.1 Representation

As explained in chapter 1, there are mainly two ways to represent multiple behaviors using HMMs:

One approach (*eg* Makris and Ellis [2002], Bennewitz et al. [2005]) is to represent all the motion patterns using a single Hidden Markov Model, where different patterns share no states and correspond to non connected components in the graph representation of the model; the other approach (*eg* Walter et al. [1999]) is to use a different HMM per motion pattern and model the belief as a mixture, where the mixing variable represents the probability that a particular HMM is the "good" model. In fact, the former solution may be seen as a special case of the latter, where the mixing variable is always uniform.

Closely related to motion pattern representation is the concept of structure, which may be regarded as defining the transition graph by choosing a number of states and selecting which state transitions are considered to be possible.

3.4.2 Learning

Learning state-space based motion pattern representations, may be decomposed in two subtasks: a) learning the structure – *ie* the topology – of the underlying transition graph; and b) learning the parameters, *ie* the transition probabilities associated with every edge of the graph, and the observation probabilities describing how observations are related to states. Sometimes also a

prior on the state is also learnt, but often it is assumed to be uniform [*eg* Walter et al., 1999, Bennewitz et al., 2005].

Structure Learning

Despite the existence of several structure learning algorithms for state-space models [*eg* Stolcke and Omohundro, 1994, Friedman, 1997, Brand, 1998], they are seldom used in the context of motion pattern learning. A notable exception is the work by Brand and Kettnaker [2000] which uses an entropic structure estimator that will be described in more detail in §4.6. Instead, structure is either fixed *a priori* or estimated using custom mechanisms.

One of such mechanisms is the use of trajectory prototype learning techniques in order to identify both the number and the form of motion patterns, and then transforming those prototypes into a state-space model. A good example is the approach proposed by Bennewitz et al. [2005], they obtain trajectory prototypes using EM as explained in §3.3, then they transform those prototypes into an HMM whose parameters are not learned, but deterministically assigned on the basis of domain knowledge and a number of assumptions about how objects move. Both [Koller-Meier and Van Gool, 2001] and [Makris and Ellis, 2002] combine trajectory prototypes and state-space models in a similar way, but the later also incorporates parameter learning into the algorithm.

Minnen and Wren [2004] present an algorithm which performs hierarchical decomposition of data to find structure at many levels of detail. The decomposition is performed by constructing a binary tree where each node contains two HMMs. The HMMs are used to classify data into two subclasses, corresponding to the node's sub trees. Classified data is sent to the respective sub tree and the bifurcation process continues until no further decomposition is possible.

An approach that integrates knowledge about the environment has been proposed by Liao et al. [2003], they use information on the static features of the environment to compute a Voronoi graph, which is then used to define the structure of a custom state-space model.

Parameter Learning

Having determined a structure, parameter learning may be performed by standard means using the Baum-Welch algorithm [Baum et al., 1970], which is a specialization of the well known Expectation Maximization algorithm [Dempster et al., 1977]. This is the case of Makris and Ellis [2002] and Liao et al. [2003]. Other authors prefer to determine parameters' values using custom procedures. Bennewitz et al. [2005] model observation probabilities as Gaussians, all having the same covariance; while transition probabilities are computed analytically under the hypothesis that objects speeds follow a Gaussian distribution.

3.4.3 Prediction

In state-space models, motion is predicted using Bayesian inference. Since the model structure is determinant in the complexity of inference, it is possible to find approaches that use either exact inference (*eg* Brand and Kettnaker [2000]), or approximate inference methods, such as particle filters [Arulampalam et al., 2002] (*eg* Walter et al. [1999], Koller-Meier and Van Gool [2001]).

3.4.4 Other state-space models

Several extensions of Hidden Markov Models have also been used for representing intentional motion, here we will provide an overview.

Pentland and Liu [1999] have proposed Markov Dynamic Models (MDM) which, conceptually are exactly like HMMs, except that the nodes of the graph represent controls (*eg* acceleration) and not the object's pose. Instead of state observations, MDM track the object with a Kalman filter and use the innovation (*ie* prediction error) of the filter as observations. Although they have shown that MDMs outperform HMMs in recognition tasks, it is difficult to use them to predict state since there is no pose representation.

Bui et al. [2002] have proposed Abstract Hidden Markov Models (AHMM), which are a hierarchical extension of HMMs that allow to represent motion at different levels of detail (*eg* metric, room, building, etc.) and integrate the concept of policy (*ie* plan), which may be regarded as the equivalent of a motion pattern; a two-level AHMM has been applied to intentional motion by Osentoski et al. [2004], using EM to learn the model's parameters. One of the advantages of AHMMs over HMMs is that they are able to provide good mid-term predictions even if they are wrong at the long-term, for example, they may accurately predict that a person is going towards the room's door even if they wrongly predict that the final destination is the kitchen. Nevertheless, unsupervised learning with AHMMs is challenging, mainly because of the difficulty of automatically finding a good (*ie* semantically sound) hierarchic decomposition of the space into regions.

All the approaches we have mentioned until now make the hypothesis that, when there is more than one object in the environment, their motion is independent, thus ignoring object interaction. This problem has been studied by Brand et al. [1997] by means of an HMM extension called Coupled Hidden Markov Models (CHMM), which represents the joint state of all objects. The main drawback of this approach is that the number of possible interactions grows exponentially with the number of interacting objects. This has motivated the work of Gong and Xiang [2003], Xiang and Gong [2006] which have proposed Dynamically-multi-linked Hidden Markov Models (DML-HMM), which are based on the idea that interactions should be modeled only when it is highly likely that they exist.

A different direction has been taken by Oliver et al. [2000], they use CHMMs to model interaction behaviors between pairs of objects (*eg* approach, meet and continue together) without modeling the environment at all. Behaviors are defined prior to learning and trained on labelled data. This approach has been proved to very successful in explaining human behavior; however, since there is no pose representation, it is not applicable to motion prediction.

3.5 Other Approaches

3.5.1 Neural network based approaches

Johnson and Hogg [1995], and Sumpter and Bulpitt [2000] have proposed similar approaches based on multilayer neural networks: the first layer is used to discretize the space by using a

Self-organizing network to perform Vector Quantization (VQ)[4] so that every unit (*ie* neuron) in this layer corresponds to a discrete state. In order to obtain the pattern representation, both approaches use an additional layer of "leaky neurons" , one for every discrete state. These leaky neurons turn "on" when an object passes through the corresponding state and their output decays slowly afterwards. As a result, the leaky neuron layer keeps a "trace" of the trajectory after the object's motion has ended; this trace (*ie* the activation values of the leaky neurons) is used to represent the learned pattern. Finally, different patterns are classified using a third layer of neurons, which performs VQ on the whole output vector of the second layer, *ie* the values of all the leaky neurons.

Hu et al. [2004a] eliminate the leaky network layer by adding lateral connections between the units in the state layer in order to account for the temporal relationships between them.

A drawback which is common to all these approaches is that they assume that the number of motion patterns to be represented is known *a priori*, which implies a fair amount of knowledge about environment. A hybrid approach, which does not share this problem, has been proposed by Stauffer and Grimson [2000]: instead of the leaky neuron layer, they represent motion patterns using a co-occurrence matrix C where every element $c_{i,j}$ corresponds to the probability that an object passes through states i and j given that it is engaged in a motion pattern.

A different neural network approach has been explored by Chang and Song [1997], they train a multilayer perceptron to predict motion on single motion patterns, obtaining good results, however, they do not deal explicitly with more than one motion pattern.

Neural network based approaches, have a performance which is comparable with that of discrete state-space models, as may be seen in the results obtained by Hu et al. [2004a]. On the downside, these approaches do not represent uncertainty in an explicit fashion, and, in the best of cases, rely in ad-hoc procedures to produce a probabilistic output.

3.5.2 Goal oriented approaches

Dee and Hogg [2004] proposed an approach which takes the intentional stance towards the motion. They model motion as being motivated by the desire of the object (*ie* the agent), to move to explicit places (*ie* goals) in the environment. The originality of the approach resides in the fact that the agent's beliefs are taken into account by modeling the world from the agent's perspective (*eg* may the agent see the object?). This makes it, to our knowledge, the only approach of this type to model the agent's perception. From there, motion is modeled on the basis of a cost model by assuming that the object will always take the cheapest path to its goal unless something happen. However, although this is interesting for scene interpretation (*ie* if the object did not take the cheapest path, then something happened), it seems less useful for motion prediction because of its relatively simple and deterministic motion model. Also, the approach assumes that the general structure of the environment is known *a priori*.

[4]This may be also regarded as performing clustering analysis. As a matter of fact, from an operative point of view, both approaches are very similar: they group data in a certain number of groups so that some error of distortion function is minimized. Their main difference is that, while cluster analysis focuses in finding clusters in multidimensional data, vector quantization aims to find the best representation of a given data set that may be obtained with a reduced number of elements.

Foka and Trahanias [2002] have also modeled motion in terms of goals, they predict motion in terms of a simple geometric method which does not take the environment into account. A similar approach has been proposed in Bruce and Gordon [2004] where the geometric method is complemented with a stochastic motion model.

3.5.3 Other

Kruse et al. [1996, 1997], Kruse and Wahl [1998] have developed a technique where motion is represented by line segments where the object is supposed to move at uniform speed and transitions do not occur between states, but between line segments.

3.6 Discussion

This chapter has provided a general review of the area of pattern based motion models. Now we would like to draw some general conclusions about existing techniques, in view of solving the main objective of this thesis: building a "learn and predict" motion prediction technique to improve autonomous navigation.

3.6.1 General issues

Here, we discuss some issues that are general to all pattern based motion models. When relevant, we will discuss how different approaches deal with them, focusing mainly in the difference between trajectory prototypes and state-space models.

Incremental learning

First of all, a "learn and predict" approach should be incremental. This means that it should verify three properties [Langley, 1995]:

- It should process data elements one by one.
- It should not reprocess any previous data element.
- It should retain only one main knowledge structure in memory.

The main reason for this requirement is efficiency. Using a batch algorithm would mean storing the complete history of observation sequences, and processing it after every trajectory has observed. This is ineffective and not well suited for continuous processing in real-time.

Representing uncertainty

In order to plan its trajectory, an autonomous vehicle needs to know in advance the state of the different obstacles that are present in the environment. At the same time, since uncertainty is inherent to prediction, a deterministic output would have little chances to be true and, at best, a

limited utility. The alternative is to explicitly represent the uncertainty by using a probabilistic representation.

Since state-based approaches are built on probabilistic models they are ideal candidates as motion prediction approaches. Trajectory prototypes, on the other hand, are deterministic models. Even when they are complemented with a variance as in [Bennewitz et al., 2002, Vasquez and Fraichard, 2004] so that they are able to model spatial uncertainty, they still fail to model the temporal uncertainty which is related to the fact that in real environments, objects not only do not move at the same speed but, most of the time, their speeds vary as they move.

Discovering new motion patterns

An incremental pattern learning approach should be able to identify when an object is involved in an unknown behavior and to create a new pattern model in consequence. This is, in general, difficult to implement for state-space models, since it involves the capacity to create dynamically new states and the valid connections between them, which is equivalent to incremental structure learning, which, as we will show in chapter 4, is a difficult problem.

Approaches based on trajectory prototypes seem to be better suited for the task. Even if most of the current approaches are designed to work off-line, it seems feasible to implement an incremental trajectory clustering approach. The same may be said about goal-oriented approaches, such as [Foka and Trahanias, 2002] and [Dee and Hogg, 2004].

Representing interaction between moving objects

A good part of human behavior involves interacting with other humans: two persons who find themselves face to face in a corridor will move in order to avoid each other, a man may go faster in order to join a friend who is walking just in front of him, etc.

But dealing with interactions between objects is complex. Modeling interactions with a state-space model implies representing the joint state of all the objects that may be simultaneously present in a scene. If we have N discrete states and M moving objects, this means learning and storing the probabilities of $O(N^M)$ possible interactions, which quickly becomes intractable.

Therefore, discrete states modeled by approaches such as [Brand et al., 1997, Oliver et al., 2000] do not represent places in the space, but a limited number – *ie* two or three – of high-level behaviors such as *avoiding* or *meeting*. This means that these techniques may not be directly used to predict the physical state of an object. However, it would be interesting to study how they may be integrated with other techniques in order to obtain such a prediction.

Besides some very general remarks in chapter 9, this problem is not addressed in this thesis.

On-board sensors

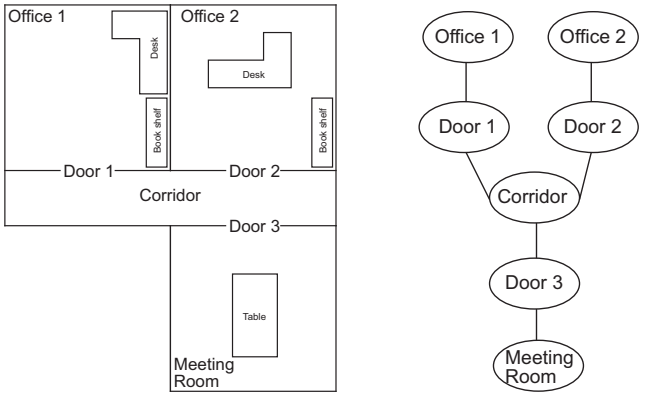
Practically all the techniques that we have reviewed in this chapter assume that there is some kind of global sensor providing information about the motion of all the objects that move in the environment. Learning and modeling intentional motion using on-board sensors is a much more difficult problem that is beyond the scope of this thesis.

3.6.2 State-space model issues

As we have mentioned before, approaches based on trajectory prototypes are not well suited to produce probabilistic state predictions. As a matter of fact, our previous works on motion prediction [Vasquez and Fraichard, 2004] were based on this kind of approaches. We were able to obtain a probability distribution representing our belief about the motion pattern that the object is actually following. But we also wanted to have a probabilistic representation of the state even when the "right" trajectory is known with certainty. The idea was to take into account the limited precision of our model as well as the differences between real trajectories which are represented by the same prototype. This ultimately lead us to the use of discrete state-space models and to the approach presented in this thesis. This section discusses some issues which are exclusive to this family of approaches.

Even when the semantics of discrete states are clear, and they are assumed to represent places in the environment, it is possible to construct different space representations which fall into two wide categories[5]: a) *metric* representations, where states represent points in the space, generally represented as a set of coordinates; and *topological* representations, where the states represent only a few interesting places which often possess a high-level semantic connotation (*eg* kitchen, corridor) and which are connected together by some notion of adjacency which is frequently represented with a graph (fig.3.5).

Even though topological representations may be seen as a particular case of metric representations – after all, a metric representation also encodes "interesting places" and their spatial relations – it is frequent to talk of a topological representation when the state variable has a low cardinality. For navigation purposes, metric representations provide more precise and useful information. At the same time, they imply a higher number of states, thus, higher computational complexity.



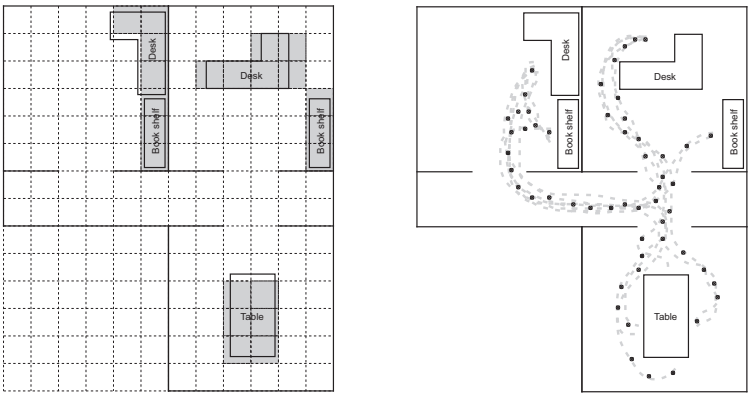
(a) Environment

(b) topological representation

Figure 3.5: A topological representation of the office environment

[5] See [Filliat, 2001] for an extended analysis of these alternatives.

A related subject is that of automatically finding a "good" space decomposition. Answering this involves a finding a compromise between precision and computational complexity, because both augment with the number of states. Thrun et al. [2005] distinguish two types of approaches to decompose the space: a) *static* decompositions such as grids, which partition the environment without taking into account how objects move in it; and b) *dynamic* approaches, which adapt to observed motion, performing finer decomposition on areas where objects move and coarser decomposition elsewhere. Although static approaches are easier to implement, they are wasteful on computational resources because they partition the space even in places where no activity takes place. Moreover, due to the "curse of dimensionality"[6], the problem worsens with the number of dimensions in the space (*eg* position, velocity, size). An example of the difference on these approaches is shown in fig.3.6.



(a) static (168 elements) (b) dynamic (54 elements)

Figure 3.6: Comparison between static and dynamic decompositions. Notice how the dynamic decomposition produces a better representation of the data (grey dots) using less than a third of the discrete elements of the grid approach.

[6] This term, first employed by Bellman [1961], refers to the exponential growth of the hypervolume of a space, when new dimensions are added; and how it demands a similar increase on the number discrete elements needed to represent the space.

Chapter 4

Hidden Markov Models

Carr. - ... your name isn't Jack, it's Tristan.
Tzara. - No, it isn't, it's Jack.
Carr. - You have always told me it was Tristan. I have introduced you to everyone as Tristan. You answer to the name of Tristan. Your notoriety at the Meieri Bar is firmly associated with the name Tristan. It is perfectly absurd saying your name isn't Tristan.
Tzara. - Well, my name is Tristan in the Meieri Bar and Jack in the library, and the ticket was issued in the library.

SIR TOM STOPPARD
Travesties

4.1 Overview

This thesis is based on the use of Hidden Markov Models as motion models, which, as we have seen in chapter 3, are probably the most popular technique in the literature of pattern based motion prediction. This chapter provides the reader with a broad introduction to this probabilistic framework. Sections 4.2 through 4.4 present what may be considered as the "classic" theory on HMMs. Readers already familiar with HMMs may safely skip these sections and proceed directly to section 4.5 and the rest of this chapter, where we discuss structure learning, a difficult problem for which no standard algorithm exists yet, in spite of the existence of several approaches in the literature.

4.2 Probabilistic Model

Hidden Markov Models (HMMs) are a specialization of the Bayes filter for discrete state variables, while it does not constrain the form of observation variables, which may be either discrete

[Rabiner, 1990] or continuous [Juang et al., 1986]. Although most of the theory for both types of observation variables is the same, some implementation differences exist (*eg* parametric forms, learning algorithm). Here, we have privileged the discussion of continuous observation HMMs because they are better suited to our particular problem.

4.2.1 Variables

Since we are going to use a local model (see §2.3.3), an HMM may be defined in terms of three variables:

- S_t, S_{t-1}, the current and previous states, which are discrete variables with value $S_t, S_{t-1} \in \{1, \cdots, N\}$ for a fixed N.
- O_t, the observation variable, which is a multidimensional vector in $\mathbb{R}^M$.

4.2.2 Decomposition

The decomposition is the same than for the Bayes filter:

$$P(S_{t-1}\ S_t\ O_t) = P(S_{t-1})P(S_t \mid S_{t-1})P(O_t \mid S_t) \tag{4.1}$$

Where the state prior is computed recursively:

$$P(S_{t-1}) = P(S_{t-1} \mid O_{1:t-1}) \tag{4.2}$$

4.2.3 Parametric forms

Hidden Markov Models make an additional conditional independence assumption with respect to the standard Bayes filter: both the observation and the transition probabilities are considered to be *stationary*, that is, independent of time:

$$P(O_i \mid S_i) = P(O_j \mid S_j) \quad \forall\ i,\ j \in \{1, \cdots, T\} \tag{4.3}$$
$$P(S_i \mid S_{i-1}) = P(S_j \mid S_{j-1}) \quad \forall\ i,\ j \in \{2, \cdots, T\} \tag{4.4}$$

This hypothesis makes it possible to define the parametric forms of the JPD probabilities without taking time into account:

- $P([S_1 = i]) = \pi_i$. The state prior is stored as a a vector $\pi = \{\pi_1, \cdots, \pi_N\}$ where each element represents the prior probability for the corresponding state.
- $P([S_t = j] \mid [S_{t-1} = i]) = a_{i,j}$. Transition probabilities are represented with a $N \times N$ *transition matrix* A where each element $a_{i,j}$ represents the probability of reaching the state j in the next time step given that the system is already in state i.

- $P(O_t \mid [S_t = i]) = \mathbf{G}(O_t;\ \mu_i,\ \sigma_i)$. The observation probability is represented by a Gaussian distribution for every state. The set of all the Gaussians' parameters are denoted as $b = \{(\mu_1, \sigma_1^2), \cdots, (\mu_N, \sigma_N^2)\}$[1].

We denote the whole set of parameters for an HMM by $\lambda = \{\pi, A, b\}$.

4.2.4 Example: the broken air conditioning system

We will illustrate the process of defining a continuous observation HMM with a simple example[2]. Let us suppose that we have an air conditioning system which may be in one of two settings: "fresh" and "cold"; due to an internal malfunction, the system switches randomly between them once a day according to the following rules:

1. If the current setting is "fresh", it may change to "cold" with probability 0.8, or stay as it is with probability 0.2.
2. If the current setting is "cold", it may change to "fresh" with probability 0.9, or stay as it is with probability 0.1.

These rules are presented as a graph in fig. 4.1:

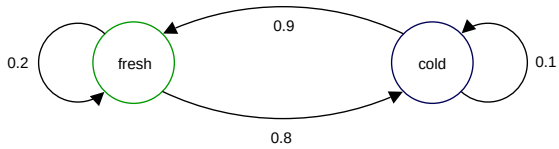


Figure 4.1: Transition graph of the air conditioning example.

Supposing that – according to its setting – the machine regulated the room temperature to two different and well defined temperatures, it would be trivial to infer directly the state by measuring their values; unfortunately, this is not the case, and the results of applying one setting or the other are rather approximative. They may be described probabilistically as follows: when the machine is in the "fresh" setting, the temperature has a mean value of 20° Celsius and a variance of 25. For the "cold" setting, the mean temperature is 15° Celsius and the variance is 16.

Representing this system as an HMM is straightforward. First, we define the domains of the state (*ie* setting) and observation (*ie* room temperature) variables:

$$D_{S_t} = \{1, 2\} \quad \text{where 1 stands for "fresh" and 2 for "cold"}$$
$$D_{O_t} = \mathbb{R}$$

[1] An extension of this idea known as mixtures of Gaussian densities, is also frequently used to approximate aribtrarily shaped observation probabilities, see [Juang et al., 1986] for further explanation.

[2] This has been inspired on an example of discrete observation HMMs which appears in [Manning and Schutze, 1999].

then, we fix the parameters' values. First we assign the state prior by assuming that, the first day, both settings are equiprobable:

$$\pi = [0.5, 0.5]$$

Transition probabilities are filled according to the specification:

$$A = \begin{bmatrix} 0.2 & 0.8 \\ 0.9 & 0.1 \end{bmatrix}$$

Similarly for the parameters of the observation probability:

$$b = \{(20, 25), (15, 16)\}$$

We have now a complete HMM definition. Of course, we would like to use it to answer questions about the behavior of the system, which is the subject of the following section.

4.3 Inference

Inference tasks on HMMs may be classified in: a) on-line inference, which takes place as the system evolves, every time that a new observation is available; and b) off-line inference, whose input consists of complete observation sequences and, therefore, is performed after the observed phenomenon has finished. The distinction is important because most off-line inference tasks are based on a dynamic programming technique called the forward-backward algorithm which will be introduced in §4.3.2.

4.3.1 On-line inference

On-line inference tasks are particularly useful for applications like navigation, where it is necessary to take decisions while objects are moving. The key to on-line inference lies in maintaining the belief state, from there, inference may be performed in terms of the local model by using the belief state as a prior and applying Bayesian inference.

State Filtering.

As explained in §2.3.3, state filtering is one of the most common probabilistic questions for Bayes filters – thus, for HMMs – the name comes from the fact that its purpose is to "filter out" observation noise in order to estimate the system state. It is done using expression (4.5), which is conceptually divided in two steps: a) *prediction* which projects the current belief state one time step into the future; and b) *update*, which improves the estimation by integrating the last obtained observation:

$$P(S_t \mid O_{1:t}) = \frac{1}{Z} \underbrace{P(O_t \mid S_t)}_{update} \underbrace{\sum_{S_{t-1}} [P(S_t \mid S_{t-1}) P(S_{t-1} \mid O_{1:t-1})]}_{prediction} \tag{4.5}$$

Prediction is the most computationally expensive step, typically, this operation involves iterating through the N possible transitions for every state, hence, its time complexity is $O(N^2)$. However, as we will show later, the complexity of the prediction step may be reduced by imposing constraints on the structure of the transition matrix.

Prediction.

Predicting the future state consists basically in, taking the belief state and then propagating it ahead in time, it may be regarded as filtering without the update step (eq. (4.6)). Given that it is necessary to iterate through every time step up to H – also known as the *time horizon* – the algorithm's complexity is $O(HN^2)$.

$$P(S_{t+H} \mid O_{1:t}) = \sum_{S_{t+H-1}} [P(S_{t+H} \mid S_{t+H-1}) P(S_{t+H-1} \mid O_{1:t})] \tag{4.6}$$

Example

Continuing with our example, let us suppose that we want to perform filtering to estimate the state of our AC machine from temperature readings. The first day, we get a temperature reading of 23.05°, so, we compute for $S_1 = 1$:

$$\begin{aligned} P([S_1 = 1] \mid [O_1 = 23.05]) &= P([O_1 = 23.05] \mid [S_1 = 1]) P([S_1 = 1]) \\ &= \frac{1}{Z_1} \mathbf{G}(23.05;\ 20,\ 25) \pi_1 \\ &= \frac{1}{Z_1} [0.066][0.5] \\ &= \frac{0.033}{Z_1} \end{aligned}$$

this is a special situation because $t = 1$ and we do not have previous observations of the system's state. Thus, instead of applying the prediction step, we just use the state prior $P([S_1 = 1])$.

Now, we compute for $S_1 = 2$:

$$\begin{aligned} P([S_1 = 2] \mid [O_1 = 23.05]) &= P([O_1 = 23.05] \mid [S_1 = 2]) P([S_1 = 2]) \\ &= \frac{1}{Z_1} \mathbf{G}(23.05;\ 15,\ 16) \pi_2 \\ &= \frac{1}{Z_1} [0.013][0.5] \\ &= \frac{0.007}{Z_1} \end{aligned}$$

substituting the value of the normalizing variable $Z_1 = 0.033 + 0.007 = 0.040$, we obtain the following probabilities.

$$P([S_1 = 1] \mid [O_1 = 23.05]) = 0.83$$
$$P([S_1 = 2] \mid [O_1 = 23.05]) = 0.17$$

These probabilities constitute the belief state for $t = 1$, which may be written as a vector:

$$\hat{S}_1 = [0.83, 0.17]$$

The second day, we measure again the temperature, now getting a reading of 17.5°. We proceed in an analogous fashion to estimate the system state for $S_2 = 1$:

$$\begin{aligned}
&P([S_2 = 1] \mid [O_1 = 23.05]\ [O_2 = 17.5]) \\
&\quad = P([O_2 = 17.5] \mid [S_2 = 1]) \sum_{S_1} P([S_2 = 1] \mid S_1) P(S_1 \mid [O_1 = 23.05]) \\
&\quad = \frac{1}{Z_2} \times \mathbf{G}(17.5;\ 20,\ 25)[a_{1,1} \times 0.83 + a_{2,1} \times 0.17] \\
&\quad = \frac{1}{Z_2} \times 0.070 \times [0.2 \times 0.83 + 0.9 \times 0.17] \\
&\quad = \frac{0.022}{Z_2}
\end{aligned}$$

and for $S_2 = 2$:

$$\begin{aligned}
&P([S_2 = 2] \mid [O_1 = 23.05]\ [O_2 = 17.5]) \\
&\quad = P([O_2 = 17.5] \mid [S_2 = 2]) \sum_{S_1} P([S_2 = 2] \mid S_1) P(S_1 \mid [O_1 = 23.05]) \\
&\quad = \frac{1}{Z_2} \times \mathbf{G}(17.5;\ 15,\ 16)[a_{1,2} \times 0.83 + a_{2,2} \times 0.17] \\
&\quad = \frac{1}{Z_2} \times 0.082 \times [0.8 \times 0.83 + 0.1 \times 0.17] \\
&\quad = \frac{0.056}{Z_2}
\end{aligned}$$

again, we substitute $Z_2 = 0.022 + 0.056 = 0.078$ to obtain the belief state:

$$\hat{S}_2 = [0.28, 0.72]$$

Notice that every filtering step requires only to process the preceding belief state, instead or iterating through all the sequence of observations from 1 up to t.

Now, we would like to predict the system's state for day 4. First, we predict for $t = 3$:

$$\begin{aligned} P([S_3 = 1] \mid O_{1:2}) &= \sum_{S_2} P([S_3 = 1] \mid S_2) P(S_2 \mid O_{1:2}) \\ &= a_{1,1} \times 0.28 + a_{2,1} \times 0.72 \\ &= 0.70 \end{aligned}$$

$$\begin{aligned} P([S_3 = 2] \mid O_{1:2}) &= \sum_{S_2} P([S_3 = 2] \mid S_2) P(S_2 \mid O_{1:2}) \\ &= a_{1,2} \times 0.28 + a_{2,2} \times 0.72 \\ &= 0.3 \end{aligned}$$

where the belief state $\hat{S}_2$ constitutes the starting point of the prediction process. Then we compute for $t = 4$:

$$\begin{aligned} P([S_4 = 1] \mid O_{1:2}) &= \sum_{S_3} P([S_4 = 1] \mid S_3) P(S_3 \mid O_{1:2}) \\ &= a_{1,1} \times 0.7 + a_{2,1} \times 0.3 \\ &= 0.41 \end{aligned}$$

$$\begin{aligned} P([S_4 = 2] \mid O_{1:2}) &= \sum_{S_3} P([S_4 = 2] \mid S_3) P(S_3 \mid O_{1:2}) \\ &= a_{1,2} \times 0.7 + a_{2,2} \times 0.3 \\ &= 0.59 \end{aligned}$$

The process continues like this for any given time horizon H.

4.3.2 Off-line inference

Off-line inference receives its name from the fact that, it answers questions about whole observation/state sequences and, in consequence, it takes place after the last observation has been collected. Examples of offline inference tasks are learning, diagnosis and classification.

Since off-line inference tasks process all the T observations in a sequence, it seems that the cost of inference should grow exponentially with respect to the length of the sequence. Fortunately, this is not the case: thanks to the use of dynamic programming, it is possible to perform exact inference in $O(TN^2)$.

The forward-backward algorithm.

This algorithm, proposed by Baum et al. [1970], applies dynamic programming techniques to avoid the redundant computations involved in repetitively applying marginalization to perform

inference (see §2.2.5). It works by precomputing two sets of probabilities (*ie* forward and backward) and storing their values in memory. Then, the probabilistic questions are answered by reformulating inference in terms of these variables. Since they are precomputed, most marginalization operations are replaced by memory accesses whose cost is constant; thus significantly reducing the number of operations required to perform inference.

Forward probabilities $\alpha_t(i) = P(O_{1:t}\ [S_t = i] \mid \lambda)$ are computed recursively using the following expression:

$$\alpha_t(i) = \left[\sum_{j=1}^{N} \alpha_t(j) P([S_t = i] \mid [S_{t-1} = j])\right] P(O_t \mid [S_t = i]) \tag{4.7}$$

This leads to an efficient algorithm (alg. 1) which computes forward probabilities sequentially, in a similar fashion to state filtering: for every time step, the algorithm iterates through all the states, and, for every state, it iterates through all its predecessors. The whole computation takes $O(TN^2)$ calculations. Fig. 4.2 illustrates the process on our example HMM.

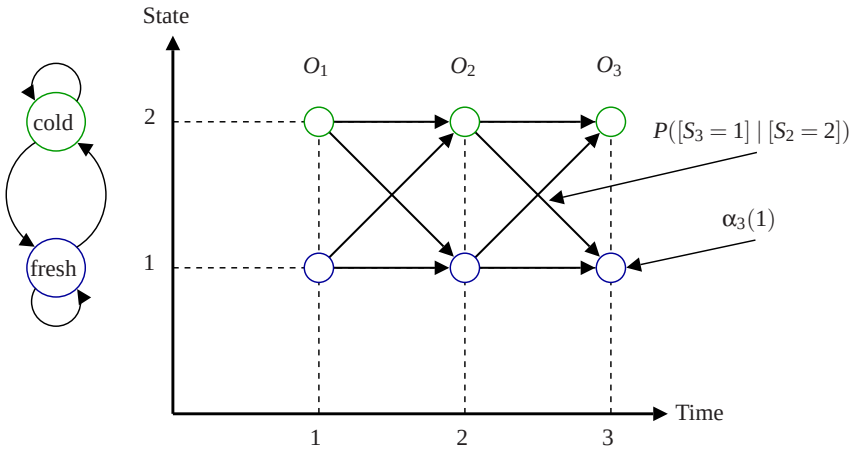


Figure 4.2: Computing forward probabilities for our example.

The use of forward probabilities to perform inference is very well illustrated by a question that appears often in HMM related problems: what is the probability of a complete observation sequence given the model parameters?[3]

A naive solution to this problem consists in marginalizing over all the state variables $S_1, \cdots, S_T$, which is equivalent to enumerating all the possible state sequences of length T:

$$P(O_{1:T} \mid \lambda) = \sum_{S_1,\cdots,S_T} P(S_1)P(S_2 \mid S_1)P(O_t \mid S_2)\cdots P(S_T \mid S_{T-1})P(O_T \mid S_T) \tag{4.8}$$

Using this equation, the computation of $P(O_{1:T} \mid \lambda)$ has an $O(N^T)$ complexity, which, in

[3] If we see the HMM as a generative model – a probabilistic automaton which produces observations – this may be interpreted as the probability that the observation sequence has been produced by the HMM.

most cases, is clearly unfeasible. Fortunately this probability may be computed much more efficiently using forward probabilities by observing that, by definition:

$$\begin{aligned} P(O_{1:T} \mid \lambda) &= \sum_{i=1}^{N} P(O_{1:T}, S_T = i \mid \lambda) \\ &= \sum_{i=1}^{N} \alpha_T(i) \end{aligned} \tag{4.9}$$

Taking into account the cost of computing all $\alpha_t(i)$, we find that $O(TN^2)$ calculations are necessary to compute the observation probability using (4.9). A huge simplification with respect to the $O(N^T)$ computations required by (4.8).

Backward probabilities $\beta_t(i) = P(O_{t+1:T} \mid [S_t = i]\ \lambda)$, are also computed recursively:

$$\beta_t(i) = \sum_{j=1}^{N} P([S_{t+1} = j] \mid [S_t = i]) P(O_{t+1} \mid [S_{t+1} = j]) \beta_{t+1}(j) \tag{4.10}$$

however, there is an important difference with respect to forward probabilities, backward probabilities depend on the *next* time step, hence, they should be computed in inverse order, starting from the last observation (which explains their name). This computation implies a problem: while, in the case of forward probabilities, the state prior is used to initialize the process, nothing similar may be done for $t = T$. The standard algorithm (alg. 2) solves this by arbitrarily setting $\beta_T(i) = 1$.

Examples of the use of backward probabilities to perform inference are *smoothing* (see next section) and parameter learning (§4.4).

Algorithm 1: *Forward_algorithm*$(O_{1:T}, \lambda)$

input : An observation sequence $O_{1:T}$
HMM parameters $\lambda = \{\pi, b, A\}$
output : Forward Probabilities $\alpha_t(i)$

begin
  for $i = 1$ *to* N **do**
    $\alpha_1(i) = P([S_1 = i]) P(O_1 \mid [S_1 = i])$
  end
  for $t = 2$ *to* T **do**
    for $j = 1$ *to* N **do**
      $\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_t(i) P([S_t = j] \mid [S_{t-1} = i])\right] P(O_t \mid [S_t = j])$
    end
  end
end
return *all* $\alpha_t(i)$

Algorithm 2: *Backward_algorithm*$(O_{1:T}, \lambda)$

input : An observation sequence $O_{1:T}$
HMM parameters $\lambda = \{\pi, b, A\}$
output : Backward probabilities $\beta_t(i)$

begin
  for $i = 1$ *to* N **do**
    $\beta_T(i) = 1$
  end
  for $t = T - 1$ *down to* 1 **do**
    for $i = 1$ *to* N **do**
      $\beta_t(i) = \sum_{j=1}^{N} P([S_{t+1} = j] \mid [S_t = i]) P(O_{t+1} \mid [S_{t+1} = j]) \beta_{t+1}(j)$
    end
  end
end
return *all* $\beta_t(i)$

Smoothing

In some situations, it is possible to obtain a better state estimation if the complete observation sequence is available. This is called *smoothing* and, as we will see, is very important for learning (§4.4).

Smoothing is computed with the forward-backward probabilities:

$$P([S_t = i] \mid O_{1:T}) = \frac{1}{p_O} \alpha_t(i) \beta_t(i) \tag{4.11}$$

Closely related to smoothing and also useful for learning is the probability of a state transition given a sequence of observations:

$$P([S_{t-1} = i]\ [S_t = j] \mid O_{1:T}) = \frac{\alpha_{t-1}(i) P([S_t = j] \mid [S_{t-1} = j]) P(O_t \mid [S_t = j]) \beta_t(j)}{p_O} \tag{4.12}$$

Viterbi decoding.

As a consequence of the state being hidden, a single sequence of observations may correspond to many different state sequences. A question that is often posed to HMMs is "what is the sequence of states which most likely corresponds to a sequence of observations?", which is also called the *best explanation* of the observation sequence. It is answered by maximizing the joint probability $\arg\max_{S_{1:T}} P(S_{1:T} \mid O_{1:T})$, that may be computed very efficiently with the *Viterbi algorithm* (alg. 3) [Viterbi, 1967, Forney, 1973].

Essentially, the Viterbi algorithm is identical to the forward algorithm except that the summation in eq. (4.7) is replaced by a maximum operation, according to the following recursion:

$$\delta_t(j) = \max_i [\delta_{t-1}(i) P([S_t = j] \mid [S_{t-1} = i])] P(O_t \mid [S_t = j]) \tag{4.13}$$

Where $\delta_t(j)$ represents the maximum likelihood of observing the partial observation sequence $O_{1:t}$ and being in state j at time t. The algorithm also stores, for every time t and state j, the preceding state $\psi_t(j)$ which has lead to j with maximum probability:

$$\psi_t(j) = \arg\max_i \{\delta_{t-1} P([S_t = j] \mid [S_{t-1} = i])\} \tag{4.14}$$

When the values for the last time step have been computed, the algorithm computes the best path by finding the state which maximizes $S_T^* = \arg\max_j \psi_T(j)$, and then backtracking according to the predecessor array: $S_{T-1}^* = \psi_T(S_T^*)$ and so on.

Algorithm 3: *Viterbi*$(O_{1:T}, \lambda)$

input : An observation sequence $O_{1:T}$
HMM parameters $\lambda = \{\pi, b, A\}$
output: Most likely state sequence $S_{1:T}^*$

begin
  for $i = 1$ *to* N **do**
    $\delta_1(i) = P([S_1 = i]) P(O_1 \mid [S_1 = i])$
    $\psi_1(i) = 0$
  end
  for $t = 2$ *to* T **do**
    for $j = 1$ *to* N **do**
      $\delta_t(j) = \max_{i \in \{1,\cdots,N\}} [\delta_{t-1}(i) P([S_t = j] \mid [S_{t-1} = i])] P(O_t \mid [S_t = j])$
      $\psi_t(j) = \arg\max_{i \in \{1,\cdots,N\}} [\delta_{t-1}(i) P([S_t = j] \mid [S_{t-1} = i])]$
    end
  end
  $S_T^* = \arg\max_{i \in \{1,\cdots,N\}} [\delta_T(i)]$
  for $t = T - 1$ *down to* 1 **do**
    $S_t^* = \psi_{t+1}(S_{t+1}^*)$
  end
  return $S_{1:T}^*$
end

Example

Returning to our example, let us suppose that we have a sequence of temperature readings for five consecutive days $O_{1:5} = \{23.05, 17.5, 18.8, 21.22, 24.32\}$ and that we need to know which sequence of states is more likely to correspond to that sequence. For this, we apply Viterbi's algorithm: first, we apply expressions (4.13) and (4.14) to compute the following table:

t	prior	1	2	3	4	5
O_t		23.05	17.50	18.80	21.22	24.32
$\delta_t(fresh)$	0.5	1.32e-2	7.42e-5	9.63e-6	5.95e-8	1.80e-09
$\delta_t(cold)$	0.5	2.62e-3	3.45e-4	1.50e-6	9.14e-8	1.25e-10
$\psi_t(fresh)$			fresh	cold	cold	cold
$\psi_t(cold)$			fresh	fresh	fresh	fresh

Then, we remark that $S_5 = fresh$ is the argument that maximizes $\delta_5(S_5)$, hence, we assign $S_5^* = fresh$. From there, we backtrack: $S_4^* = \psi_5(fresh) = cold$, and so on down to $t = 1$. At the end, we obtain the following maximum likelihood sequence:

$$S_{1:5}^* = \{fresh, cold, fresh, cold, fresh\}$$

Classification

Another useful question consists in choosing between several HMMs, representing different classes of state sequences. This is a common situation, for example, in speech recognition tasks, where every word is encoded as a different HMM.

Classification consists in finding the HMM which is more likely to correspond to a given observation sequence. Often, this is done by applying a maximum likelihood criterion, which means that the decision is taken using the output of eq. (4.9) to select the model. Nevertheless, other classification criteria may also be applied, we will discuss them in §4.6.

4.3.3 Numerical stability and HMM scaling

When implementing HMMs in computers, numerical precision problems may become an issue during the execution of the Viterbi and forward-backward algorithms. The reason is that these algorithms multiply very long sequences of numbers which are smaller than 1; and the small values that result may exceed the machine representation capabilities. This situation may be observed in our example for the Viterbi algorithm (§4.3.2), where values become appreciably small, even for a very short sequence of observations.

Solutions to this problem vary according to the algorithm:

Viterbi Algorithm. Since the viterbi algorithm operates on the probability maxima for each state, the problem is solved by using log probabilities. Hence, multiplications become additions, which are much easily handled by standard machine floating point representations.

Forward-backward Algorithms. Whereas the forward and backward algorithms perform both multiplications and sums on probabilities, applying logarithms is not a viable solution. Several solutions exist [*eg* Minka, 1999, Mann, 2006], perhaps the most popular is the one described by Rabiner [1990], which consists in multiplying forward probabilities by *scaling coefficients* in order to keep them in the dynamic range of the machine. These coefficients are computed by normalizing the forward probabilities for every time value:

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{4.15}$$

So that the scaled forward probabilities are:

$$\bar{\alpha}_t(i) = c_t \alpha_t(i) \tag{4.16}$$

The same set of coefficients that have been computed for α_t are also used to scaled Backward probabilities:

$$\bar{\beta}_t(i) = c_t \beta_t(i) \tag{4.17}$$

4.4 Parameter Learning

Let us have learning data in the form of K observation sequences $O = \{O^1, \cdots, O^K\}, O^k = \left\{O_1^k, \cdots, O_{T_k}^k\right\}$. Supposing that the state is observable, it is straightforward to obtain a maximum likelihood estimate of the model's parameters by counting (cf. §2.2.7):

$$\pi_i = \frac{\sum_{k=1}^{K} C([S_1^k = i])}{\sum_{k=1}^{K} \sum_{n=1}^{N} C([S_1^k = n])} \qquad \forall\, i \in \{1, \cdots, N\} \tag{4.18}$$

$$\mu_i = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} C([S_t^k = i])\, O_t^k}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} C([S_t^k = i])} \qquad \forall\, i \in \{1, \cdots, N\} \tag{4.19}$$

$$\sigma_i^2 = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} C([S_t^k = i])\, (O_t^k - \mu_i)^2}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} C([S_t^k = i])} \qquad \forall\, i \in \{1, \cdots, N\} \tag{4.20}$$

$$a_{i,j} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k - 1} C([S_t^k = i]\, [S_{t+1}^k = j])}{\sum_{k=1}^{K} \sum_{t=1}^{T_k - 1} C([S_t^k = i])} \qquad \forall\, i,\ j \in \{1, \cdots, N\} \tag{4.21}$$

Unfortunately, state is hidden, which means that counting may not be used. The standard solution is a specialization of the Expectation-Maximization algorithm [Dempster et al., 1977] known as the Baum-Welch algorithm.

4.4.1 The Baum-Welch Algorithm

This algorithm, originally introduced by Baum et al. [1970] to estimate parameters on a single sequence of discrete observations, was later extended for multiple observation sequences by Levinson et al. [1983] and for continuous observations by Juang et al. [1986].

The basic idea of the Baum-Welch algorithm is to estimate $P(S_{1:T} \mid O^k)$ using inference and to use the expected event counts as Expected Sufficient Statistics ESS to obtain a new estimate of the model's parameters. This may be regarded as replacing the count functions that appear in eq. (4.18) to (4.21) by their corresponding probabilities, inferred from the current model

Algorithm 4: *Baum_Welch*$(O^{1:K}, \lambda)$

input : A set of observation sequences $O^{1:K}$
Initial estimate of HMM parameters λ
output : HMM parameters $\lambda = \{\pi, b, A\}$

1 **begin**
2 $converged = false$
3 **while** *not converged* **do**
4 Compute forward (α_t^k) and backward (β_t^k) probabilities as well as probability p_{O^k} of every observation sequence O^k
5 **for** $i \in \{1, \cdots, N\}$ **do**
6 $\bar{\pi}_i = \frac{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \alpha_1^k(i)\ \beta_1^k(i)}{K}$
7 $\bar{\mu}_i = \frac{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=1}^{T_k} \alpha_t^k(i)\ \beta_t^k(i)\ O_t^k}{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=1}^{T_k} \alpha_t^k(i)\ \beta_t^k(i)}$
8 $\bar{\sigma}_i^2 = \frac{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=1}^{T_k} \alpha_t^k(i)\ \beta_t^k(i)\ (O_t^k - \mu_i)^2}{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=1}^{T_k} \alpha_t^k(i)\ \beta_t^k(i)}$
9 **for** $j \in \{1, \cdots, N\}$ **do**
10 $\bar{a}_{i,j} = \frac{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=2}^{T_k} \alpha_{t-1}^k(i) P([S_t = j | [S_{t-1} = i]\ \lambda) P(O_t^k | [S_t = j]\ \lambda) \beta_t^k(j)}{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=2}^{T_k} \alpha_{t-1}^k(i) \beta_{t-1}^k(i)}$
11 **end**
12 **end**
13 **if** $\lambda = \{\bar{\pi}, \bar{A}, \bar{b}, \}$ **then**
14 $converged = true$
15 **end**
16 **else**
17 $\lambda = \{\bar{\pi}, \bar{A}, \bar{b}, \}$
18 **end**
19 **end**
20 **end**

parameters λ and observation sequence O^k. For the sake of efficiency, inference is performed using forward-backward probabilities.

Since expected counts and model parameters are interdependent, the computation is iterated. As shown by Baum et al. [1970] the algorithm is guaranteed to converge to a local maximum of data likelihood.

Analysis

The algorithm makes extensive use of the forward-backward probabilities in lines 6 through 10:

- Line 6 computes the state prior using the expected counts, which are calculated using

(4.11):

$$\bar{\pi}_i = \frac{\sum_{k=1}^{K} P([S_1 = i] \mid O^k\ \lambda)}{\sum_{k=1}^{K} \sum_{S_1} P(S_1 \mid O^k\ \lambda)} = \frac{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \alpha_t^k(i)\ \beta_t^k(i)}{K} \tag{4.22}$$

- Lines 7 and 8 also use the expected counts, obtained with (4.11) to compute the observation probability mean values and covariances:

$$\bar{\mu}_i = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} P([S_t = i] \mid O^k\ \lambda)\ O_t^k}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} P([S_t = i] \mid O^k\ \lambda)} = \frac{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=1}^{T_k} \alpha_t^k(i)\ \beta_t^k(i)\ O_t^k}{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=1}^{T_k} \alpha_t^k(i)\ \beta_t^k(i)} \tag{4.23}$$

$$\bar{\sigma}_i^2 = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} P([S_t = i] \mid O^k\ \lambda)\ (O_t^k - \mu_i)^2}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} P([S_t = i] \mid O^k\ \lambda)} = \frac{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=1}^{T_k} \alpha_t^k(i)\ \beta_t^k(i)\ (O_t^k - \mu_i)^2}{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=1}^{T_k} \alpha_t^k(i)\ \beta_t^k(i)} \tag{4.24}$$

- Line 10 computes the expected transition counts using (4.12) an uses it to estimate transition probabilities:

$$\begin{aligned} \bar{a}_{i,j} &= \frac{\sum_{k=1}^{K} \sum_{t=2}^{T_k} P([S_{t-1} = i]\ [S_t = j] \mid O^k \lambda)}{\sum_{k=1}^{K} \sum_{t=2}^{T_k} P([S_{t-1} = i] \mid O^k \lambda)} \\ &= \frac{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=2}^{T_k} \alpha_{t-1}^k(i) P([S_t = j \mid [S_{t-1} = i]\ \lambda) P(O_t^k \mid [S_t = j]\ \lambda) \beta_t^k(j)}{\sum_{k=1}^{K} \frac{1}{p_{O^k}} \sum_{t=2}^{T_k} \alpha_{t-1}^k(i) \beta_{t-1}^k(i)} \end{aligned} \tag{4.25}$$

Since the forward-backward probabilities are computed only once, and then stored in a table, the cost of computing the state prior and observation probabilities if $O(N|O|)$ where $|O|$ is the total number of observations in training data; in a similar way, the cost of computing the transition probability is $O(N^2|O|)$. However, these costs may be considerably reduced by restricting the model's structure, which is the subject of sections 4.5 and 4.6.

4.4.2 Incremental algorithms

A number of incremental versions of the Baum-Welch algorithm exist in the literature. Neal and Hinton [1998] propose a simple modification, called incremental EM, which, instead of summing over all the observation sequences in the data set in lines 6 through 4.25, processes only one sequence at a time. This is justified by the fact that, since the initial values of the expected counts are assumed to be inaccurate, it is best to update them as soon as possible. The algorithm stores the sum of the expected counts for all the probability distributions, and, in the case of HMMs, it is guaranteed to converge to a local maxima of data likelihood.

Another approach has been proposed by Singer and Warmuth [1996]. Instead of storing expected counts and maximizing the data likelihood $P(O_{1:T_k}^k \mid \lambda^t)$, they maximize an alternative objective function $F(\lambda)$, which penalizes large changes in parameters:

$$F(\lambda^{t+1}) = \lambda P(O^k_{1:T_k} \mid \lambda^t) - d(\lambda^t, \lambda^{t+1}) \tag{4.26}$$

where $d(\lambda^t, \lambda^{t+1})$ is a measure of the difference between the current and the reestimated parameters.

4.5 Transition structure

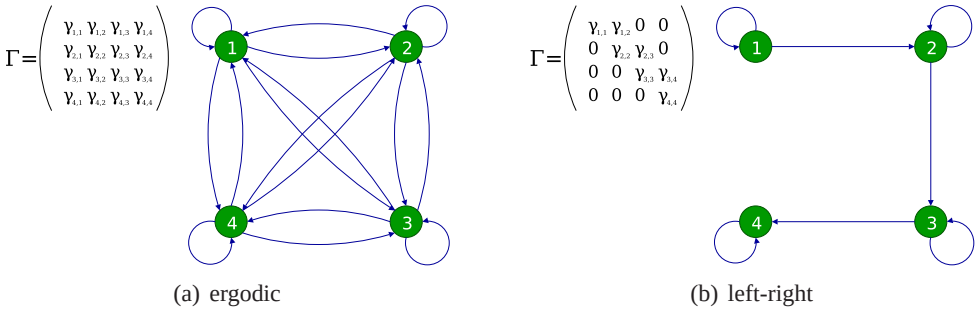
Until now, in all our complexity computations for inference and learning algorithms, we have considered a fully connected or *ergodic* HMM (fig. 4.3(a)) where every state may be reached from any other state in a single time step. As we have seen in §4.3.1, this implies the need to iterate through N^2 possible transitions when computing the prediction step. For some applications, however, a better representation may be built by assuming that certain transitions are not possible. This is equivalent to constraining the structure of the transition matrix by forcing the value of a subset of the $a_{i,j}$ elements to be zero. This is often called the *structure* or *topology* of the HMM, and, even if they are somewhat related, it should not be confused with the structure of a Bayes network.

More formally, we define the structure Φ of an HMM as the specification of the number of states N and the subset of elements in A such that $a_{i,j} = 0$. In other words, Φ specifies the cardinality of the state variable and a subset of forbidden transitions and leaves all other transitions unspecified (but constrained to be strictly positive). A complete model specification consists of the model's structure and parameters $\mathcal{M} = \{\Phi, \lambda\}$.

One of the most popular HMM structures are called left-right models (fig. 4.3(b)). Their fundamental property is that their transitions coefficients obey the following constraint:

$$a_{i,j} = 0 \quad \forall j < i \tag{4.27}$$

It is easier to visualize the HMMs structure as a graph (fig. 4.3), where nodes represent states and for every $a_{i,j} \neq 0$ there is a directed edge from i to j indicating that that transition is allowed. In this case, state j is said to be a *neighbor* of state i.



(a) ergodic (b) left-right

Figure 4.3: Examples of HMM topologies. Both the transition matrix and the corresponding graph are shown.

Of course, other topologies than ergodic or left-right models may be imagined. An interesting case is when A is composed primarily by zeroes (*ie* sparse). In those cases, it is more efficient to represent the transition matrix as an array of linked lists, one list per state (*ie* a transition list). Each linked list stores the neighbors of the corresponding state (fig. 4.4). This representation is not only more compact, it allows to compute the prediction step of filtering in $O(E)$ where E is the number of non-zero elements in the transition matrix.

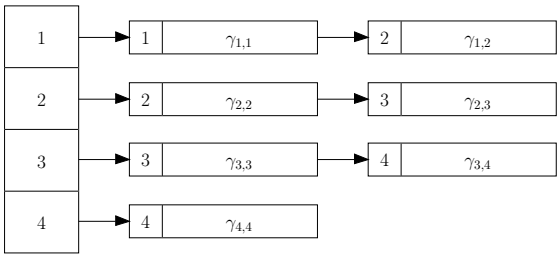


Figure 4.4: A linked list representation of fig. 4.3(b)

The performance gain may be illustrated by an example. Let us have a 5×5 grid, where every cell represents a state. By only allowing transitions between cells having a common edge [4], we reduce the number of possible transitions from $125^2 = 15625$ for an ergodic model to just 80 for the 4-neighborhood criterion (fig. 4.5).

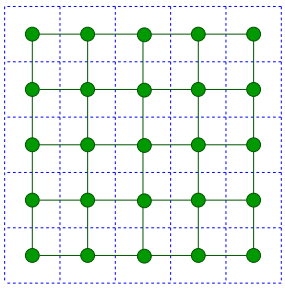


Figure 4.5: A 5×5 grid (dotted blue lines) and the corresponding non directed topology graph (solid green lines) assuming 4-neighborhood.

Defining a non-ergodic topology for the model does not only lead to increased time and memory efficiency, but also influences the quality of inference [cf. Brand, 1998, Freitag and McCallum, 2000, Binsztok and Artières, 2005] and accelerates learning by reducing the number of parameters. But, how to choose the best structure for a given application or problem? It seems natural to do the same as for the models parameters and to learn the structure from data, which is the topic of the following section.

[4] This is called 4-neighborhood, because every cell which is not in the border has exactly four neighbors under this criterion

4.6 Structure Learning

A good starting point to illustrate structure learning is to consider the full Bayesian learning problem, which consists in computing the posterior probability of the model given the data:

$$P(\mathcal{M} \mid O) = \frac{P(O \mid \mathcal{M})P(\mathcal{M})}{P(O)} \tag{4.28}$$

But $P(\mathcal{M} \mid O)$ gives the joint posterior of the structure and the parameters, and we are interested in the posterior probability of the structure alone. Thus, we marginalize over the space of all possible parameters:

$$P(\Phi \mid O) = \frac{P(\Phi)P(O \mid \Phi)}{P(O)} \tag{4.29}$$

$$= \frac{P(\Phi)\int_{\lambda} P(\lambda \mid \Phi)P(O \mid \Phi\, \lambda)d\lambda}{P(O)} \tag{4.30}$$

Which gives us a probability distribution over all the possible structures. However, computing the full probability distribution is almost never done in practice. Instead, learning is performed by finding a Maximum *a Posteriori* (MAP) estimate of the structure, *ie* the single model that maximizes eq. (4.30).

Although it is tempting to simplify further and apply a Maximum Likelihood criterion by assuming that $P(\mathcal{M})$ is uniform, Maximum Likelihood (ML) estimation suffers from a bias towards complex models which renders it unusable for structure learning[5]: it is trivial to construct a maximum likelihood structure by representing every observation in O with a different discrete state.

Hence, it is necessary to maximize eq. 4.30, which is difficult, because no closed form solution for the integral is known Murphy [2002]; a common workaround is to use an approximation of the integral's value known as the Bayes Information Criterion (BIC) [Schwarz, 1978], which is expressed as a log-likelihood:

$$\log P(O \mid \Phi) \approx BIC(O, \Phi, \hat{\lambda}) = \log P(O \mid \Phi\, \hat{\lambda}) - \frac{d}{2}\log|O| \tag{4.31}$$

Where $\hat{\lambda}$ is the maximum likelihood estimate of the parameters given the current structure – obtained, for example, using the Baum-Welch algorithm; d is the number of free parameters in the model; and $|O|$ is the number of observations in training data. The first term is just data likelihood (4.9) and the second one may be understood as a penalty for model complexity. The BIC converges asymptotically to the log data likelihood as $|O|$ grows.

Taking logarithms in (4.30) and replacing $\log P(O \mid \Phi)$ by the BIC, we obtain:

$$\log P(\Phi \mid O) \approx \log P(\Phi) + BIC(O, \Phi, \hat{\lambda}) - \log P(O) \tag{4.32}$$

[5] Maximum likelihood has also been subject to more fundamental critics concerning its semantic and mathematical soundness [Irony and Singpurwalla, 1997]

Hence, the learning problem becomes that of maximizing (4.32) instead of (4.30). In general, the term $\log P(O)$ is also excluded from the maximization because it does not depend on the structure. A further simplification is to assume a flat prior for the structure $P(\Phi)$, given that complexity is already penalized by the BIC. Thus, in many approaches, just the BIC is maximized.

Maybe the most important difficulty in structure learning lies in the huge size of the space of structures. Even for a fixed number of states N, the number of possible adjacency matrices is $O(2^{N^2})$. This makes it necessary to restrict the structure to a smaller family of graphs (*eg* trees, directed acyclic graphs) [*eg* Pearl, 1988, Meila et al., 2001]; an alternative is to conduct an heuristic based search in structure space.

4.6.1 Local search algorithms

Due to the complexity of structure estimation, instead of using a global algorithm to maximize (4.32), a local search is often conducted, where the algorithm starts with an initial guess of Φ and then searches in its *neighborhood* for structures having a higher BIC. Neighbors are obtained by applying simple operators to Φ such as adding, deleting or reversing an edge (fig. 4.6). Depending on the algorithm, one or more of those structures are selected and the process is iterated.

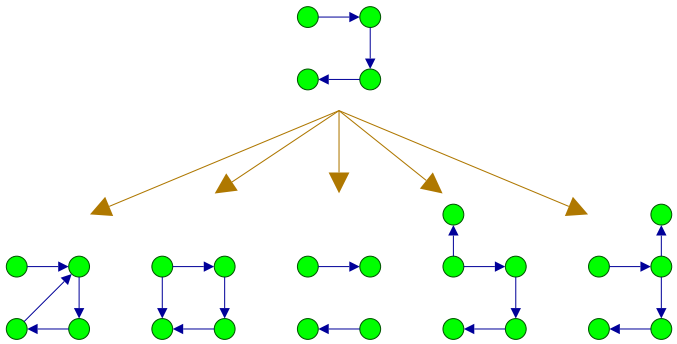


Figure 4.6: Local structure search: neighbors are found by adding or removing nodes and edges.

Despite the use of approximations like the BIC, local search strategies are still expensive, mainly because the EM (Baum-Welch) algorithm must be run for every candidate structure. This has motivated a different approach proposed by Friedman [1997]: instead of running the EM algorithm inside the search procedure, the search is performed inside EM (alg. 5). This is known as *Structural EM*.

The key to structural EM lies in steps 7 and 8. The estimation of $\hat{\lambda}'$ is performed by updating $\hat{\lambda}$ according to the difference between Φ and Φ', which, as we mentioned above, consists of a small change such as and added or deleted edge. Hence, it is not necessary to run the whole Baum-Welch algorithm for every neighbor, but only for the one that gets the best BIC score in every iteration. The algorithm is guaranteed to converge to a local maximum of the BIC score.

Algorithm 5: *Structural_EM*$(O^{1:K}, \lambda, \Phi)$

input : A set of observation sequences $O^{1:K}$
Initial estimate of HMM parameters λ
Initial structure estimate Φ
output : HMM parameters $\lambda = \{\pi, b, A\}$
HMM Structure Φ

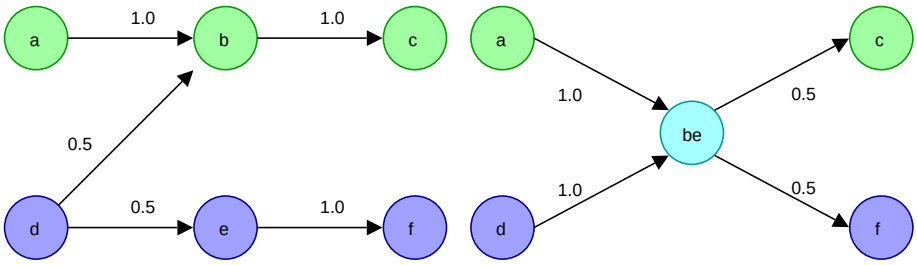
begin
  $converged = false$
  while *not converged* **do**
    Improve λ using EM
    for *each neighbor* Φ' *of* Φ **do**
      Compute expected counts for Φ' using Φ and λ ; `/* Structural E-Step */`
      Compute $\hat{\lambda}'$ using the expected counts
      Compute $BIC(O^{1:K}, \Phi', \hat{\lambda}')$
    end
    $\Phi^* = \arg\max_{\Phi'} BIC(O^{1:K}, \Phi')$
    if $BIC(O^{1:K}, \Phi^*, \hat{\lambda}') > BIC(O^{1:K}, \Phi, \lambda)$ **then**
      $\Phi = \Phi'$; `/* Structural M-Step */`
      $\lambda = \hat{\lambda}'$
    end
    else
      $converged = true$
    end
  end
end

4.6.2 State merging algorithms

State merging algorithms [Stolcke and Omohundro, 1994, Seymore et al., 1999, Binsztok and Artières, 2005] work by applying heuristics which are very similar to agglomerative clustering. First, they assume that every observation in O corresponds to a different state, and assigns a probability of one to edges connecting states corresponding to consecutive observations and zero to all others. The result of this step is a maximum likelihood model, but which is overly complex and do not generalize at all.

From there, the algorithm builds a more simple – and general – model by merging states (see fig. 4.7). At every iteration, the algorithm searches for the merging which gives the highest increase on the posterior (4.30) and it stops when no further increase is detected.

In order to reestimate the model parameters, model merging uses a simplified version of the Baum-Welch algorithm, which only reestimates the parameters for the most probable path – which is computed using the Viterbi algorithm – instead than for the whole model; thus reducing computation time at the expense of optimality. However, experimental results seem to be



(a) before (b) after

Figure 4.7: Merging states *b* and *e* produces state *be*. Notice how the probabilities associated to incoming edges are summed, while those associated to outgoing edges are renormalized.

comparable to those obtained using Baum-Welch.

Model merging is able to estimate the number of states N in an intuitive fashion. Moreover, it seems possible to adapt it to work incrementally. On the downside, model merging is too sensible to differences in temporal alignment and it has been mostly tested on applications having discrete observations.

4.6.3 Other algorithms

Another interesting approach has been proposed by Brand [1998], instead of searching or building the structure, he transforms the Baum-Welch algorithm into a MAP estimation procedure called Entropic-EM by incorporating an entropic prior in the computation of the new parameters. This prior favors informative values (those that are near from zero or one) when training data is scant, but converges to the conventional maximum likelihood estimation estimate as more observations are available. The result of using this MAP criterion is that irrelevant parameters are driven asymptotically to zero. After Entropic-EM learning, edges or states which are considered not informative are trimmed from the structure.

A somewhat similar idea has been proposed by Vasko et al. [1997]. They first train a fully connected HMM using Baum-Welch. From there, they iteratively remove transitions or states from the model. Since the fully connected model has maximum data likelihood, every iteration decreases it. The chosen structure is the simplest one before a substantial decrease in the likelihood.

The idea of starting with a very simple model to which new nodes and transitions are added during learning – hence, incrementing data likelihood – has also been explored by a number of authors Lockwood and Blanchet [1993], Freitag and McCallum [2000], but these approaches are not general, since they all apply domain-specific knowledge.

4.7 Discussion

In this chapter we explained the basic concepts regarding Hidden Markov Models. We presented the probabilistic model: variables, JPD and parametric forms, making emphasis in con-

tinuous observation HMMs. We also presented the "classic" inference and learning algorithms for HMMs, showing that maintaining the belief state constitutes the basis for inference. Correspondingly, we illustrated the use of dynamic programming to perform efficient off-line inference. We outlined the use of the Baum-Welch algorithm, which, basically consists in using off-line learning to compute the ESS which are necessary to estimate the model's parameters. Finally, we have introduced the problem of structure learning, and presented an overview of the existing approaches.

We would like to conclude with some reflections about the use of HMMs in the context of pattern based motion prediction. As we have seen in chapter 3, HMMs are very popular in the literature of pattern based motion prediction, but, what are the reasons of this popularity? indeed, there are many reasons: First, HMMs are a discrete model, which makes them simpler to manipulate than a continuous representation. Moreover, the existence of efficient and well tested algorithms for inference and learning makes their application straightforward.

On the other hand, in spite of the efficiency of existing algorithms, the computational cost of inference on conventional ergodic Hidden Markov Models grows with the square of the number of discrete states in the model; which makes it too complex to scale their use to problems which require hundreds or even thousands of states and huge data sets.

There exist at least three ways to address this problem: a) using hierarchical representations such as Abstract Hidden Markov Models Bui et al. [2002] which decompose the problem using a divide and conquer strategy; b) defining simpler transition structures which, at the same time reduce the algorithmic complexity and provide a better representation of motion; and c) using approximate inference tools like particle filters [Arulampalam et al., 2002].

The problem becomes even harder when learning gets involved. Even in the off-line case, learning is difficult for hierarchical approaches because they rely on semantic information (*eg* doors, rooms, buildings) which is considerably hard to learn with an unsupervised algorithm. On the other hand, approximate inference tools are not necessary, provided that learned structures are simple enough to allow real-time inference.

Recapitulating, for a working "learn and predict" approach based on HMMs, we want a learning algorithm that satisfies the following conditions:

1. Both parameters and structure should be learned.
2. Learning should be incremental.
3. The learnt structure should be both meaningful and simple enough to allow inference in real-time.

Despite the existence of incremental extensions of the Baum-Welch and of structure learning algorithms, for the best of our knowledge there exists only one algorithm – state merging – which is able to simultaneously learn the parameters *and* the structure of an HMM in an incremental way. Unfortunately, there is no guarantee that the structures obtained by applying model merging will be simple and semantically sound. Therefore, in order to fulfill the third condition, a new learning algorithm is needed, which is the subject of chapter 5.

Part III

Proposed Approach

Chapter 5

Growing Hidden Markov Models

> It's all to do with the training: you can do a lot if you're properly trained
>
> QUEEN ELIZABETH II
> *Television documentary*

5.1 Overview

As we have shown during the first part of this thesis, Hidden Markov Models constitute a powerful probabilistic tool. Nevertheless, in order to be able to apply them to the problem studied in this thesis, it is necessary to device incremental parameter and structure learning algorithms which are able to work in real time. Moreover, as we have explained at the end of chapter 4, we require the learnt structures to be both simple and meaningful.

This chapter introduces our proposed solution – and the main contribution of this thesis – Growing Hidden Markov Models (GHMM). They may be described as time-evolving HMMs with continuous observation variables, where the number of states, topology and probability parameters are updated every time that an observation sequence is available.

Our approach focuses in the utilization of HMMs as an approximate inference tool for continuous state spaces. We assume that the continuous state space is discretized into a finite number of regions, and that every such region is represented by a discrete state in the HMM. Second, we assume that state evolves continuously. A third assumption is that observations produced by states which are near from each other are also near from each other. It is important to note that, although this assumptions somewhat restrict the applicability of our approach; they are shared by a large number of problems, at least in robotics [cf. Thrun et al., 2005].

The key intuition behind GHMMs, is that the structure of the HMM should reflect the spatial structure of the state space discretization, where transitions between states are only allowed if the corresponding regions are neighbors. Hence, structure learning consists basically in estimating the best space discretization from data and identifying neighboring regions. We have addressed this problem by building a *topological map* of the environment.

For parameter learning, we basically have adapted the approach proposed by Neal and Hinton [1998] in order to deal with variable state cardinality and continuous observations.

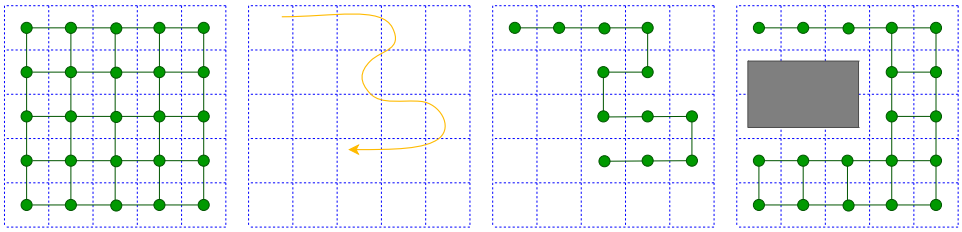
The following section introduces the notion of topological map and explains its relationship with the HMM structure. Then we will discuss Vector Quantization and Topology Representing Networks, in order to introduce the concrete algorithm which we have used, the Instantaneous Topological Map. Next, we will provide a formal description of Growing Hidden Markov Models, explaining how the topological map is integrated into the incremental structure and parameter learning algorithm.

5.2 The topological map

The topological map is a discrete representation of the state-space in the form of a graph, where nodes represent discrete regions and edges indicate that the regions which correspond to the linked nodes are contiguous, *ie* it is possible to move continuously between them without passing through any other region[1].

We will illustrate the concept with a simple grid discretization of a two dimensional space (fig. 5.1(a)). The corresponding topological map will have one node for every cell in the grid, and edges between cells that share a border.

The state is assumed to evolve in a continuous – in the mathematical sense – fashion in space. Thus, if we plot its trajectory, it should inevitably pass through cells borders as it goes from one discrete region to another (fig. 5.1(b)). Since, by definition, there is one edge for every border, it follows that the best approximation (fig.5.1(c)) to any possible continuous trajectory may be built using only edges that are already in the graph[2].



(a) grid and graph (b) trajectory in space (c) trajectory in the graph (d) an obstacle

Figure 5.1: a) Example of a discretization of space into a grid of rectangular cells (blue dotted lines), and the corresponding topological map representation (green dots and solid lines); b) a continuous trajectory; c) the same trajectory represented as a succession of nodes and edges in the map; and d) an obstacle (gray rectangle) and the corresponding topological map.

[1] It is important to note that, for us, a topological map is essentially a metric representation in which neighborhood is represented explicitly. This differs from the concept of a topological map we have discussed in §3.6.2, which have a coarser representation often based on higher level semantics [*eg* Kuipers, 1998]

[2] For the sake of simplicity, in this example we assume it is not possible to move in diagonal passing exactly through a corner.

Often the process evolves in a small manifold of the state space. For example, moving objects tend to pass frequently through some regions (*eg* corridors) while completely bypassing others (*eg* obstacles). This may not be captured by grids and other static space decompositions, because they are chosen in advance. Therefore, the whole space is represented, with the consequent waste of computational resources (see §3.4).

It would be preferable to discretize space according to observed data in order to approximate the actual manifold in which state evolves so that "forbidden" regions of the space are not represented (see fig. 5.1(d)). This poses the problem of how to perform this discretization and, at the same time, identify neighbor regions in an efficient way. Moreover, the solution should be incremental.

Fortunately, there exists a family of tools which deals precisely with this problems: Topology-representing networks (TRN) [Martinetz and Schulten, 1991]. They incrementally build a topological map by iterating through two steps a) partition space in discrete regions using Vector Quantization and b) find pairs of neighbor regions and link their respective centers. They are the subject of the next section.

5.3 Vector Quantization and Topology Representing Networks

Vector quantization is a data compression technique originally developed in the context of signal processing but widely used in very diverse technical domains. The idea of vector quantization is to encode a continuous D-dimensional input data manifold M by employing a finite set $C = \{c_1, \cdots, c_K\}$ of reference D-dimensional vectors. A point x of the manifold is represented using the element of C which is closest to it according to a given distance measure $d(x, c_i)$, such as the square error, or the Euclidean distance.

This procedure induces an implicit partition of the manifold in a number of subregions

$$\mathcal{V}_j = \{x \in M \mid d(x - c_j) \leq d(x - c_i) \forall i\} \tag{5.1}$$

called Voronoi regions, such that every input vector that is inside a Voronoi region $\mathcal{V}_j$ is described by the corresponding reference vector c_j.

The goal of a vector quantization algorithm is to find values for the reference values in order to minimize the mean quantization error, also known as the *distortion*:

$$E = \sum_{i=1}^{K} \int_{x \in \mathcal{V}_i} d(x, c_i) P(x) dx \tag{5.2}$$

Since, in most cases, the form of the manifold is unknown, the error may not be computed directly, instead, it is estimated from a data set consisting of $|X|$ samples, or input vectors:

$$\hat{E} = \frac{1}{|X|} \sum_{i=1}^{K} \sum_{x_j \in \mathcal{V}_i} d(x_j, c_i) \tag{5.3}$$

which assumes a uniform prior over input vectors.

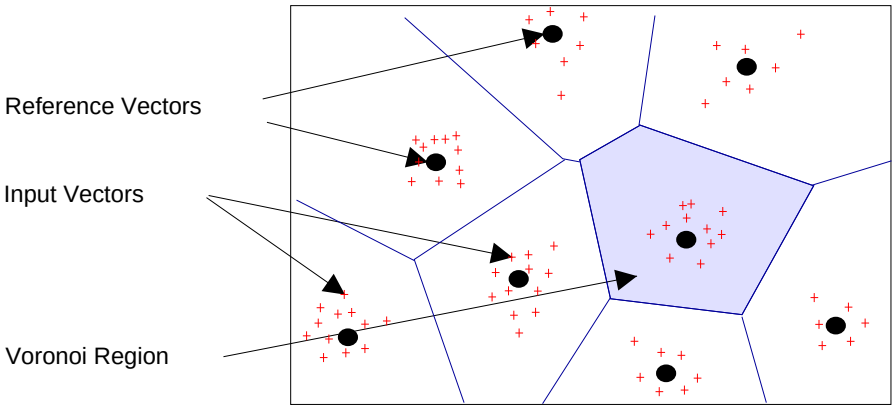


Figure 5.2: Example of a partition in Voronoi regions: there are some 2-dimensional input vectors (red crosses). Reference vectors are represented by big points and Voronoi regions are indicated by blue boundary lines, the set of all those boundaries is called a Voronoi graph.

The most widely used vector quantization algorithm is the k-means or Lloyd algorithm [Lloyd, 1957, Linde et al., 1980], which may be regarded as a version of the Expectation-Maximization algorithm, where a hard ownership criterion is used (*ie* every data element is assigned to one cluster only).

Although standard k-means is guaranteed to converge to a local minimum of the distortion, it has a number of drawbacks: a) the number of reference vectors to be found should be known *a priori*; b) the quality of the output is highly dependent on initialization; c) the fact that it is a batch algorithm makes it too expensive for very big data sets.

Even if the last drawback has been addressed by MacQueen [1967], who proposed an incremental version of the algorithm – known as on-line k-means – the other two problems remain difficult and have motivated the research of alternative vector quantization approaches.

One of such approaches, which is particularly interesting, is the use of Kohonen Networks, or Self-organizing map (SOM) [Kohonen, 1995]. The main difference between the SOM and other classic vector quantization algorithms is that, in the SOM, there are links between reference vectors (or units, as they are known in the SOM jargon) so that they form a network. Links describe a neighborhood relationship between units and define a topology over the entire network. This topology is defined *a priori* to form a chain – for a one dimension Kohonen map – or a grid – for the two dimensional case.

In addition to vector quantization, the SOM learning algorithm ensures that, after learning, points that are close in the input data manifold M will be associated to the same unit, or to units which are close in the chain or grid. Hence, network links effectively encode additional information – called a topographical mapping – on the similarity of the represented input data. This additional information is useful in applications such as motion planning [Krose and Eecen, 1994], and speech processing [Kohonen, 1988]. Moreover, SOMs have been shown to reduce the initialization problem of k-means [Bacao et al., 2005].

However, Kohonen networks introduce a new problem: to obtain an optimal topographical

mapping and minimize the quantization error, it is necessary that the topology of the network matches that of the represented manifold. In the case of Kohonen networks this requires prior knowledge about the topological structure of the manifold, which is often unavailable. This problem has motivated the proposal of a new family of approaches known as Topology Representing Networks, which are able to learn the manifold's structure while performing vector quantization.

5.3.1 Topology Representing Networks

Topology Representing Networks are based on the idea of connecting neighbor Voronoi regions – *eg* regions with a common border – with edges, called delaunay's edges. The set of all edges constitutes the dual of the Voronoi graph and is known as the Delaunay's triangulation (fig. 5.3).

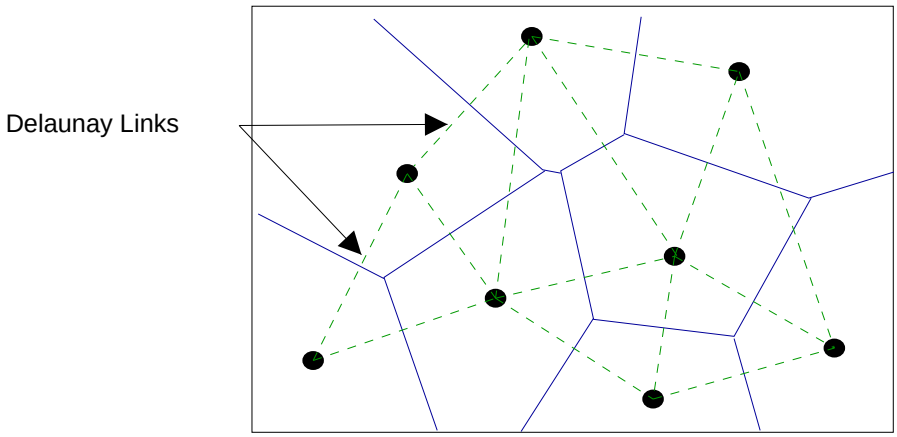


Figure 5.3: Voronoi graph (blue) and delaunay triangulation (green).

TRNs represent the network's topology with a subset of the Delaunay's triangulation. The edges are learned by using the competitive hebbian rule – also known as hebbian learning – proposed by Martinetz and Schulten [1991], which consists in creating a new edge between two units every time that, for a given input, they are the two closest units to that input and they are not already linked (fig. 5.4).

Although the first TRNs in the literature [*eg* Martinetz and Schulten, 1991]. worked only with a fixed number of units, later approaches, such as the Growing Neural Gas (GNG) [Fritzke, 1995], and the Grow When Required (GWR) Networks [Marsland et al., 2002], evolved as adaptive structures which are able to insert or even delete units during learning.

A common feature of these TRNs is that they adapt reference vectors by applying constant factors (*ie* learning rates) meaning that convergence to a minimum of distortion is not guaranteed, but this is not necessarily a drawback, since it allows the network to learn permanently. Indeed, this characteristic allows the network to show an adaptive behavior, since it is able to "forget" what it has learned in the past.

The advantages and drawbacks of adaptive TRNs, when compared to conventional k-means

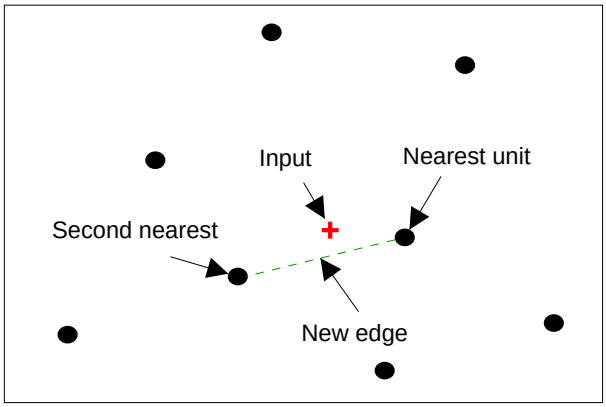


Figure 5.4: Hebbian learning

or EM vector quantization approaches may be summarized as follows:

Advantages

1. They use incremental learning algorithms, thus, they may be applied to very large data sets or to on-line learning from data streams.

2. They do not require any prior knowledge on the number of reference vectors.

3. Due to their growth mechanism, which uses observation data to initialize new state representations, they are quite robust to initialization condition problems.

4. They are able to learn the data manifold's topology, even when it is mixed (*eg* it contains one, two and three dimensional submanifolds).

5. They are assumed to be a plausible model of how biological entities process information [cf Martinetz and Schulten, 1991].

Drawbacks

1. They do not converge to a minimum of distortion (5.3) but, at best, oscillate continuously around one.

2. Lacking or incomplete formal theories about convergence and stability.

From our point of view, the drawbacks of TRN are largely compensated by their advantages, which explains our decision of using a TRN – the Instantaneous Topological Map – as the basis for our learning algorithm. It will be described in detail in the following section.

5.4 The Instantaneous Topological Map

We have chosen the Instantaneous Topological Map (ITM) of Jockusch and Ritter [1999] as the basis of our algorithms based on two reasons. First, the ITM algorithm – as opposed to GNG and GWR – is designed from the beginning to deal with data which is correlated in time, such as trajectories. The second reason is that the algorithm has a reduced set of parameters having clear physical meaning, moreover, no prior knowledge about the topology or the size of the state-space are required in order to select the algorithm's parameters.

The algorithm works on the basis of a distance measure, which, in our case, is the Mahalanobis distance, instead of the more classic Euclidean distance.

We will now provide some basic definitions before going into a detailed description of the algorithm.

5.4.1 Definitions

The ITM algorithm builds incrementally a set $\mathcal{U}$ of nodes, and a set $\mathcal{L}$ of edges connecting nodes. The input of the algorithm consist of input vectors which, in the context of this thesis, we identify with the observations O_t which constitute the input of an HMM.

Associated with every node i there is a reference vector or weight w_i, as described in §5.3.

An edge between nodes i and j will be denoted as (i, j). Moreover, edges are not directed, thus, it holds that $(i, j) \equiv (j, i)$. A useful concept is the *neighborhood* of a node i, which is the set of all nodes to which i is linked:

$$\mathcal{N}(i) = \{ j \in \mathcal{U} \mid (i, j) \in \mathcal{L} \} \tag{5.4}$$

Last, we will introduce the Mahalanobis distance which will be used as distance criterion in the algorithm. It is defined in terms of a covariance matrix Σ as:

$$d_\Sigma^2(u, v) = (u - v)^T \Sigma^{-1} (u - v) \tag{5.5}$$

We have preferred its use over the Euclidean distance because of the following reasons:

1. It permits to take into account the correlations between the different dimensions of state space.
2. It is more general than Euclidean distance, and includes it as a particular case.
3. It is scale-invariant.
4. Since the observation probabilities are Gaussians, it makes sense to take into account the Gaussian's covariance to build the topological map.

5.4.2 Algorithm

The ITM algorithm has three main goals: minimizing the distortion, finding the number of nodes N, and finding the edges that define the topology $\mathcal{L}$. The algorithm has only three parameters:

Algorithm 6: *ITM-Update*$(O_t, \Sigma, \tau, \varepsilon : \mathcal{U}, \mathcal{L})$

input :
Input vector O_t
Covariance matrix Σ
Insertion Threshold τ
Smoothing factor ε

modifies:
Topological map nodes $\mathcal{U}$
Topological map edges $\mathcal{L}$

begin
  $b = \arg\min_{i \in \mathcal{U}} \; d^2_\Sigma(w_i, O_t)$; /* Determine the best unit */
  $s = \arg\min_{i \in \mathcal{U} \setminus b} \; d^2_\Sigma(w_i, O_t)$; /* Determine the second best unit */
  $w_b = w_b + \varepsilon(O_t - w_b)$; /* Weight adaptation */
  if $s \notin \mathcal{N}(b)$ *and* $d^2_\Sigma(wb, w_s) < 4\tau$ **then**
    $\mathcal{L} = \mathcal{L} \cup \{(b,s)\}$; /* Create the link */
  end
  for $i \in \mathcal{N}(b)$ **do**
    $\bar{w}_{b,i} = (w_i + w_b)/2$
    if $d^2_\Sigma(\bar{w}_{b,i}, w_s) < d^2_\Sigma(\bar{w}_{b,i}, w_i)$ *and* $d^2_\Sigma(w_b, w_i) > 1$ **then**
      $\mathcal{L} = \mathcal{L} \setminus (b,i)$; /* Delete link (b,i) */
      if $\mathcal{N}(i) = \emptyset$ **then** $\mathcal{U} = \mathcal{U} \setminus i$; /* Remove node i as well */
    end
  end
  $\bar{w}_{b,s,=}(w_s + w_b)/2$
  if $(d^2_\Sigma(\bar{w}_{b,s}, w_s) < d^2_\Sigma(\bar{w}_{b,s}, O_t)$ *or* $d^2_\Sigma(w_s, O_t) > 4\tau)$ *and* $d^2_\Sigma(w_b, O_t) > \tau$ **then**
    $\mathcal{U} = \mathcal{U} \cup \{r\}$; /* Create a new node */
    $w_r = O_t$
    if $d^2_\Sigma(w_b, w_r) < 4\tau$ **then** $\mathcal{L} = \mathcal{L} \cup \{(b,r)\}$; /* Connect nodes r and b */
    if $d^2_\Sigma(w_b, w_s) < \tau$ **then** $\mathcal{U} = \mathcal{U} \setminus s$; /* Remove s */
  end
end

Covariance Matrix (Σ). It is used to compute the Mahalanobis distance.

Insertion Threshold (τ). It may be regarded as the average radius of a discrete region, in terms of the Mahalanobis distance.

Smoothing Rate (ε). It regulates the rate at which reference vectors are adapted.

The algorithm starts having two connected nodes, whose weights may be initialized at random or, from example, taking the values of the two first observations. The core of the algorithm is the update procedure, which adapts the network for a single observation O_t, the procedure may be conceptually separated in four steps:

Matching. Steps 2 and 3 find the best and second best nodes, with respect to the Mahalanobis distance. The covariance matrix is assumed to be the same for all nodes. This is the most expensive step of the algorithm, because of its dependence on the number of nodes. Nevertheless, since the dependency is linear, the algorithm is well suited form most real-time demands even for a large number of nodes.

Weight adaptation. Step 4 smooths the learnt weights according to the smoothing coefficient ε. Jockusch and Ritter mention that this parameter may be set to zero without affecting the overall performance of the network in a significant way.

Edge adaptation. Steps 5 through 14 perform competitive hebbian learning to insert a new Delaunay edge between the best and the second best units. Here we have made a slight modification of the normal ITM algorithm by allowing edge creation only if it is relatively short. If, after the creation of the new edge, other edges become invalid Delaunay, they are deleted, eventually leading to the deletion of nodes with no remaining edges. This contrasts with other TRNs, which rely on ad-hoc procedures to destroy invalid Delaunay edges.

Node adaptation. Steps 15 through 21 are responsible of inserting a new node when the observation is considered to be ill-represented. As a consequence of node creation, the second best node may be deleted if it is redundant, that is, if it is too close to b (fig. 5.5). As in the case of edge adaptation, we have slightly modified the original ITM algorithm to limit the length of the edges created in this step.

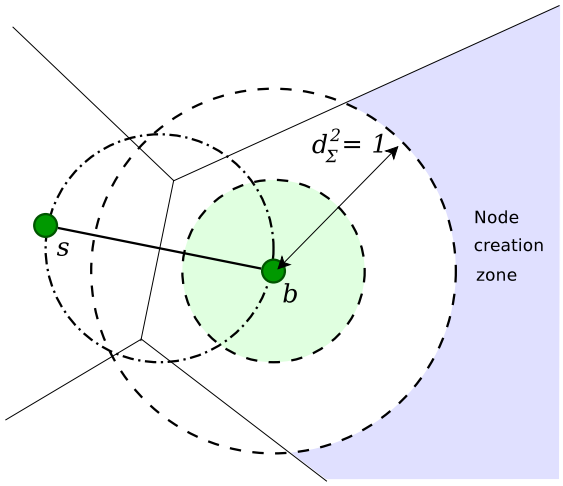


Figure 5.5: ITM Node adaptation. A node is created when it falls in the light blue zone, this may be followed by the deletion of the second best node if it inside the light green zone.

The combined effect of edge and node adaptation is that network nodes are uniformly arranged in such a way that the average distance between two neighboring nodes is τ. This imposes

a limit on the minimum quantization error that may be achieved by the ITM algorithm. Indeed, the covariance and insertion threshold together may be regarded as describing the maximum resolution, or precision which may be attained while discretizing the space. This also means that the number of nodes in the network will depend on Σ and τ.

It should be noted that, since node distribution is regular, the reference vectors learned by the ITM algorithm are not disseminated according to the observation prior, which is a well known property of the GNG algorithm, where node density is roughly proportional to $P(O_t)$. At the same time, this does not constitute a particularly desirable property when approximating continuous state spaces; in fact, often the most informative states are also least probable, thus, it makes sense to represent different regions of space with the same precision, in spite of the relative probability of observing something inside them, as long as this probability is greater than zero.

5.4.3 Properties

Convergence

One theoretical drawback of the ITM algorithm is that strict convergence to a local minimum of the distortion is not verified. The reason is that the algorithm, like other Topology Representing Networks, uses a constant learning rate, which leads to models which, instead of converging, keep oscillating around local minima.

However, in the case of ITM it is possible to establish a theoretical upper limit to the distortion, based on the fact that the algorithm guarantees that the maximum Mahalanobis distance to an observation will be equal or inferior to τ.

From (5.3):

$$\hat{E} = \frac{1}{|O|} \sum_{i=1}^{K} \sum_{O_t \in \mathcal{V}_i} d_{\Sigma}^2(O_t, w_i) \tag{5.6}$$

where $|O|$ is the number of input vectors or observations in training data. But, since every observation belongs to only one Voronoi region and $d_{\Sigma}^2(O_t, w_i) \leq \tau$, we know that:

$$\hat{E} \leq \tau \tag{5.7}$$

Number of edges

As we have stated at the beginning of this chapter, we want the learnt structure – thus, the topological map – to be both simple and meaningful. We have seen that the topological map is a subset of the Delaunay triangulation, which has a simple interpretation in terms of neighborhood, and seems plausible when partitioning a continuous space into discrete regions.

But, what about simplicity? it turns out that the upper bound on the number of edges is the same than for the Delaunay triangulation, which, for reference vectors in $\mathbb{R}^2$, the number of edges is $O(N)$ because the triangulation's edges and vertices form a planar graph [Aurenhammer, 1991]. However, for higher-dimensional spaces things seem less promising, for example Dewdney and Vranch

[1977] have proved that the upper bound for the size of a Delaunay triangulation in $\mathbb{R}^3$ is $O(N^2)$, which is equivalent to a fully connected graph.

On the other hand, Dwyer [1989] has demonstrated that the expected size of the Delaunay triangulation in any d-dimensional space is $O(N)$, supposing that the reference vectors are drawn uniformly from the unit sphere. This result indicates that high-dimensional Delaunay triangulations will be small ($O(N)$) in most practical situations [cf. Aurenhammer and Klein, 2000].

Complexity

The ITM algorithm is designed from the ground up for incremental learning. It processes observations in a one by one basis. The time complexity of the update algorithm is $O(N)$. Moreover, since the most expensive operation is the matching step, it seems feasible to improve the efficiency of the algorithm by using a hierarchical space indexing techniques, like R-trees and their extensions [*eg* Guttman, 1984, Beckmann et al., 1990].

5.5 Probabilistic Model

As we have seen, the ITM algorithm, is able to produce find a simple and semantically sound topological map by incrementally processing observations. This seems to fulfill most of our requirements for the structure.

In this section we explain, how our algorithm integrates the ITM algorithm to learn the structure of the HMM. We will also explain how we have modified parameter learning in order to accommodate for an ever evolving structure. As in the case of the HMM, we will present the approach as a probabilistic model.

5.5.1 Variables

The only difference at this level between GHMMs and HMM is that the domain of the state variable changes as time passes. Besides that, both models use the same variables:

- S_t, S_{t-1}, defining the current and previous states, respectively. The domain of both variables varies with time and is represented by $\mathcal{S}_k$, which is the set of discrete states in the GHMM structure after k observation sequences have been processed.

- O_t, which describes the current observation. Observations are assumed to be continuous vectors in $\mathbb{R}^D$, where D is the dimensionality of the continuous state space.

5.5.2 Decomposition

The form of the JPD is also the same than for hidden Markov models:

$$P(S_{t-1}\ S_t\ O_t) = P(S_{t-1})P(S_t \mid S_{t-1})P(O_t \mid S_t) \tag{5.8}$$

5.5.3 Parametric forms

Although the parametric forms are basically same than for HMMs, the way that parameters are stored is different in GHMMs:

- $P(S_t)$. Like in HMMs, the state prior is represented by a multinomial probability distribution. However, instead of directly storing probabilities in the parameter vector π, we will store the ESS for computing those probabilities. In this case, they consist of the cumulated sum of the expected counts (see §4.4). This means that $P(S_t)$ should be now computed in terms of the ESS by normalizing the counts:

$$P(S_t = i) = \frac{\pi_i}{\sum_{S_t} \pi_{S_t}} \tag{5.9}$$

- $P(O_t \mid S_t)$. All observation probability Gaussians are assumed to have the same covariance Σ. Hence, the observation probability parameters consists exclusively of the Gaussian mean values $b_i = \mu_i$ and the observation probability is computed with:

$$P(O_t \mid S_t = i) = \mathbf{G}(O_t;\ \mu_i,\ \Sigma) \tag{5.10}$$

- $P(S_t \mid S_{t-1})$. As for the state prior, instead of storing transition probabilities directly, we store the ESS as the cumulated sum of the expected counts in A. Hence, a normalization is also needed to compute $P(S_t \mid S_{t-1})$:

$$P(S_t = j \mid S_{t-1} = i) = \frac{a_{i,j}}{\sum_{S_{t-1}} a_{S_{t-1},S_t}} \tag{5.11}$$

The reason for storing the cumulated sums in A and π is to allow the incremental computation of the probabilities. Supposing that probabilities were stored directly, it would be necessary to take into account the fact that stored probabilities represent all of the already processed observation sequences, while the current expectation of the number of counts has been computed on the basis of just one – the last – observation sequence. Thus some kind of decreasing learning rate would be necessary to achieve convergence.

5.6 Inference

Since GHMMs are HMMs, they may answer the same probabilistic questions (see §4.3). For convenience, we will review in this section the two basic questions that should be answered in the context of motion prediction: filtering and prediction.

Filtering is performed in order to update the belief state of an object on the basis of an observation. This is done using expression (4.5) which we copy here:

$$P(S_t \mid O_{1:t}) = \frac{1}{Z} P(O_t \mid S_t) \sum_{S_{t-1}} [P(S_t \mid S_{t-1}) P(S_{t-1} \mid O_{1:t-1})]$$

Where $P(S_{t-1} \mid O_{t-1})$ is the belief state calculated in the previous time step.

Having estimated the current state, the future state H time steps ahead from the present is computed with:

$$P(S_{t+H} \mid O_t) = \sum_{S_{t+H-1}} P(S_{t+H} \mid S_{t+H-1}) P(S_{t+H-1} \mid O_t) \tag{5.12}$$

5.7 Structure and Parameter Learning

The main difference between conventional HMMs and GHMMs lies in the learning algorithm (see alg. 7), which iterates through two steps: a) updating the topological map of the space; and b) using the Baum-Welch mechanism to update the parameters of the state prior and the transition probability. Regarding the incremental aspect of the algorithm, it is worth noting that the input of the algorithm consists of complete observation sequences, thus, learning is performed only when such a sequence is available and not for every observation.

The algorithm maintains a topological map consisting of a node list ($\mathcal{U}$), an edge list ($\mathcal{L}$) and the node weights ($\mathcal{W}$). From the map, observation probabilities are obtained by using the node weights as the mean values of the Gaussians and assuming a fixed covariance Σ. Then transition structure is updated from the ITM edge list, and the parameters of the transition and state prior probabilities are recomputed using the sum of the expected counts[3].

The learning algorithm has the following parameters:

State prior counter default value (π_0). This value is used to initialize the state prior table when a new state (node) is created, it works as a "pseudocount", much in the same way that Dirichlet priors (see §2.2.7).

Transition counter default value (a_0). It is analogous to π_o but it applies to transition probabilities.

Covariance Matrix (Σ). All the Gaussians which define the observation probability are assumed to have the same covariance matrix. It is also this covariance that is used for the ITM algorithm.

Insertion Threshold (τ). This is one of the parameters of the ITM algorithm, it determines – together with the covariance – the resolution of the discretization, as described in §5.4

Smoothing Rate (ε). As explained in §5.4, ε determines the rate of at which node weights adapt to new data.

The algorithm is decomposed in three parts:

Topological map update. Lines 2 - 4 use the ITM algorithm as a subroutine to update the topological map. It should be noted that later steps of the algorithm (lines 5- 19) will need

[3] Indeed, this may be seen as an application of incremental EM to learn the prior and transition probabilities. The interested reader is referred to [Neal and Hinton, 1998] for a more thorough discussion of this technique.

to know which nodes and edges have been created and removed; hence it is necessary to modify *ITM_update* to store these in a temporary structures. An interesting observation is that, although this step is relatively expensive ($O(TN)$) it is possible to update the map immediately after receiving every observation. Moreover, if learning is performed in parallel with filtering or prediction, the code may be interleaved in the state filtering cycle, thus, further reducing computation time.

HMM topology update. Lines 5 through 19 update the GHMM structure to reflect that of the topological map. It is interesting to note that, if a linked list (cf. §4.5) is used to represent the transition matrix A, then assigning 0 to $\gamma_{i,j}$ is equivalent to deleting the corresponding element from the list, which enhances both memory and time efficiency. An important remark is that, when a new state is created, a self-transition is added to the structure. This self transition, allows representing situations in which the state does not changes between consecutive time steps. This does not necessarily mean that the continuous state of the object has not changed, but only that it is still in the same discrete region than in the preceding time step.

Parameter update. The observation probability parameters are updated by copying the weights of topological map to b. On the other hand, π and A are updated using forward-backward probabilities to compute the expected values as in Baum-Welch.

5.7.1 Learning the covariance

One possible extension of the learning algorithm would be to learn the individual covariance matrices of the observation probabilities. As a matter of fact, efficient incremental algorithms exist to perform this computation [*eg* Li et al., 2003, Igel et al., 2006]. However, since the covariances are used to compute the Mahalanobis distance, using theses algorithms would modify the ITM algorithm, with the following consequences:

- Since the distance criterion would not be uniform for all the space, it is not sure that the properties of the ITM algorithm will still hold. Hence, further mathematical analysis would be needed to ensure it.

- Due to the fact that the observations that have been already associated to a node in the ITM determine its covariance matrix and, thus, the dimensions of its Voronoi regions, it is possible to fall in a situation where the covariance matrix becomes progressively smaller as time passes.

Some measures may be taken to tackle this problems, for example, it is possible to use a fixed covariance matrix for the ITM algorithm and to learn different covariances for the HMM. However, due to time limitations, we were not able to perform a deeper exploration of these alternatives.

5.8 Discussion

This chapter presented a novel extension of HMMs: Growing Hidden Markov Models, which mainly differs from the standard technique in the fact that the model structure is not fixed, but evolves continuously as more observations are available. Our approach is applicable to those cases in which a Hidden Markov Model is used as a discrete approximation to model a continuous state process.

The main insight behind GHMMs is that, since discrete states are obtained by discretizing the state-space in regions, allowed transitions should be allowed only between neighbor regions. Hence, the problem of structure learning becomes that of building a topological map, by discretizing the continuous state space into discrete regions and identifying neighboring regions. In our approach, this is done by applying the Instantaneous Topological Map, a Topology Representing Network which is well suited for correlated data such as trajectories. By integrating the ITM algorithm with incremental parameter learning, we have been able to build a learning approach which fulfills the three requirements that we have defined at the end of chapter 4:

1. The algorithm learns both the parameters and the structure of the HMM.
2. The algorithm is incremental.
3. The learnt structure has an intuitive meaning, and it is simple enough to allow exact inference in real time.

Having presented GHMMs, we are now ready to explain how they may be used in the context of motion prediction, which is the subject of the following chapter.

Algorithm 7: *HMM-Update*$(O_{1:T}, \Sigma, \tau, \varepsilon : \mathcal{U}, \mathcal{L}, \mathcal{W}, \lambda)$

input :
Observation sequence $O_{1:T}$
Covariance matrix Σ
Insertion Threshold τ
Smoothing factor ε

modifies:
Topological map nodes $\mathcal{U}$
Topological map edges $\mathcal{L}$
Topological map weights $\mathcal{W}$
HMM parameters $\lambda = \{\pi, b, A\}$

begin
  for $t \in \{1, \cdots, T\}$ **do** /* Update the ITM */
    $EnhancedITM\_update(O_t)$
  end
  for *every new node* $i \in \mathcal{U}$ **do**
    $\pi_i = \pi_0$; /* initialize prior */
    $a_{i,i} = a_0$; /* initialize self-transitions */
  end
  for *every node i that has been removed from* $\mathcal{U}$ **do** /* remove priors */
    $\pi_i = 0$
  end
  for *every new edge* $(i, j) \in \mathcal{L}$ **do** /* initialize transitions */
    $a_{i,j} = a_0$
    $a_{j,i} = a_0$
  end
  for *every edge* (i, j) *that has been removed from* $\mathcal{L}$ **do** /* remove transitions */
    $a_{i,j} = 0$
    $a_{j,i} = 0$
  end
  for $i \in \mathcal{U}$ **do** /* update mean values from the ITM */
    $\mu_i = w_i$
    $\sigma_i = \Sigma$
  end
  Precompute forward (α_i), backward (β_i) and joint observation probabilities (p_O) for the observation sequence $O_{1:T}$
  for $i \in \mathcal{U}$ **do** /* incremental Baum-Welch */
    $\pi_i = \pi_i + \frac{\alpha_t(i)\ \beta_t(i)}{P_O K}$
    for $j \in \mathcal{N}(i)$ **do**
      $a_{i,j} = a_{i,j} + \frac{\sum_{t=2}^{T} \alpha_{t-1}(i) p([S_t=j]|[S_{t-1}=i]\ \lambda) p(O_t|[S_t=j]\ \lambda) \beta_t(j)}{\sum_{t=2}^{T} \alpha_{t-1}(i) \beta_{t-1}(i)}$
    end
  end
end

Chapter 6

Learning and Predicting Motion with GHMMs

> – Would you tell me, please, which way I ought to go from here?
> – That depends a good deal on where you want to get to – said the Cat.
> – I don't much care where– said Alice.
> – Then it doesn't matter which way you go – said the Cat.
>
> LEWIS CARROLL
> *Alice's adventures in wonderland*

6.1 Overview

In chapter 5, we have presented GHMMs from a general perspective. In contrast, this chapter focuses on the application of GHMMs as a tool which we use to predict the motion of pedestrians and vehicles. Therefore, the issues addressed here – as well as their respective solutions – are specific to this application and they probably may not be generalized to other domains.

Our application is based on the assumption that people and vehicles move in function of their intention to reach a particular state (*ie* its goal): a car moves in a car park in order to stop at a parking place, a person walks in an office with the intention to reach his desk, etc. Accordingly, we model the object's motion in terms of an extended state vector which is composed of two set of variables describing its *current* and *intended* states, respectively.

An important difference between our approach and other HMM based techniques is that motion patterns are not defined in terms of typical trajectories. Instead, they are determined by the object's goal: two objects are involved in the same motion pattern if they intend to reach the same state. Under this interpretation, a motion pattern may be interpreted as the set of paths which lead to a given goal[1]. As a consequence of this interpretation, the representation of motion

[1] Indeed, our representation bears some resemblance to Markov Decision Processes [Howard, 1960,

patterns is no longer restricted to chains as illustrated in fig. 6.1.

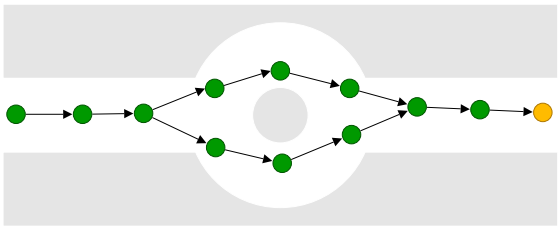


Figure 6.1: Example of a motion pattern defined in terms of a goal (orange node)

This chapter is organized as follows: in section 6.2, we explain how the extended state integrates with GHMMs to represent motion patterns. Section 6.3 discusses the use of our model to predict motion, and how, thanks to the extended state, it is possible to predict the state that an object intends to reach as a side effect of updating the belief state. In section 6.4 we describe the particularities of learning in the context of this application. In section 6.5 we illustrate our approach using an example. Then, in section 6.6, we compare our approach against existing HMM based approaches. Finally, we present our concluding remarks for this chapter in section 6.7.

6.2 Probabilistic Model

Strictly speaking, our proposed motion model is a standard GHMM which may be manipulated using the techniques that we have already described in 5. However, our definition of an extended state in terms of the current object's state and its goal[2] has implications that need to be analyzed in detail.

Until now, we have assumed that observations coming from sensors are available at every time step. In this application, this assumption is not always true: while it still holds for the current object state, it is no longer true in the case of goals.

In effect, by definition, the goal that an object aims to reach may not be observed – either directly or indirectly – until the object has reached it, that is, when its trajectory has ended. On the other hand, this situation does not arise in the case of learning because it takes complete trajectories as input, meaning that the last observation for every trajectory is known and, therefore, that goal observations (*ie* the last observation of a sequence) are available for the learning algorithm.

This section describes how GHMMs are used to build motion models using our extended state definition, it also details how to cope with goal observations, which, as we have shown, are only available at the end of an observation sequence.

Cassandra et al., 1996], a popular probabilistic planning tool.

[2]Hereafter, we will use the terms "intended state" and "goal" interchangeably. Also, unless noted otherwise, we will assume that the current and intended states are points in space, nevertheless, it should be noted that other state definitions are also possible (*eg* position, velocity, size).

6.2.1 Variables

In the context of this application, we would like to consider our probabilistic model at two different levels. At the higher, more general level, our model behaves just like a conventional GHMM and the extended state is not different from other state definitions. At a lower level, we want to distinguish between the current and the intended state, thence, we will decompose the observation variable into a current component – denoted as O'_t – and an intended, or goal component – denoted as O''_t. This results in the following variables:

- S_t, S_{t-1}, defining the current and previous *augmented* states, respectively. These variables take values in $\mathcal{S}_k$, which, as described previously, is the set of model's states after k trajectories have been processed.
- O_t, which describes the current observation. Observations are assumed to be continuous vectors in $\mathbb{R}^{2D}$, where D is the dimensionality of the standard state variable. As explained above, observations are decomposed in two components representing the current and the intended state: $O_t = [O'_t, O''_t]$.

6.2.2 Decomposition

At the higher level, the JPD decomposition is the same as for GHMMs:

$$P(S_{t-1}\ S_t\ O_t) = P(S_{t-1})P(S_t \mid S_{t-1})P(O_t \mid S_t) \tag{6.1}$$

However, at the lower level, observation variables actually represent the joint occurrence of their current and intended components, which implies that observation probability may be rewritten in terms of these components.

$$P(O_t \mid S_t) = P(O'_t\ O''_t \mid S_t) \tag{6.2}$$

we will assume that the current and intended components of the observations are conditionally independent given the current state:

$$P(O'_t\ O''_t \mid S_t) = P(O'_t \mid S_t)P(O''_t \mid S_t) \tag{6.3}$$

which permits us to rewrite the JPD in terms of the components of the observation variable:

$$P(S_{t-1}\ S_t\ O'_t\ O''_t) = P(S_{t-1})P(S_t \mid S_{t-1})P(O'_t \mid S_t)P(O''_t \mid S_t) \tag{6.4}$$

6.2.3 Parametric forms

The parametric forms are essentially the same that for GHMMs (see chapter 5), with some added restrictions for the covariance of the observation probability:

- $P(S_t)$. Will be represented as a multinomial, computed on the basis of the sum of expected counts stored in a vector π.

- $P(O_t \mid S_t)$. Due to our conditional independence assumption, the observation probability is written as a product of probabilities $P(O'_t\ O''_t \mid S_t) = P(O'_t \mid S_t)P(O''_t \mid S_t)$. Let us define those probabilities as:

$$P(O'_t \mid [S_t = i]) = \mathbf{G}(O'_t;\ \mu'_i,\ \Sigma') \tag{6.5}$$

and:

$$P(O''_t \mid [S_t = i]) = \begin{cases} \mathbf{U}_{O''_t} & \text{if } O''_t \text{ is not available} \\ \mathbf{G}(O''_t;\ \mu''_i,\ \Sigma'') & \text{otherwise} \end{cases} \tag{6.6}$$

where μ'_i and μ''_i are the mean values of the current state and the goal for state i; and Σ' and Σ'' are the respective values of the covariance matrix for all the states.

By noting that $P(O_t \mid S_t)$ is either a product of Gaussians, or a product of a constant and a Gaussian, we may it write this probability as a single Gaussian:

$$P(O_t \mid [S_t = i]) = \frac{1}{Z}\mathbf{G}(O_t;\ \mu_i,\ \Sigma) \tag{6.7}$$

where $\mu_i = [\mu'_i, \mu''_i]$, and Σ is a a block diagonal matrix[3] having the form:

$$\Sigma = \begin{bmatrix} \Sigma' & 0 \\ 0 & \Sigma'' \end{bmatrix} \tag{6.8}$$

And Z is a normalization variable, which permits to compute the uniform on the goal component using the same Gaussian representation. This is done by assigning the same value (*eg* zero) to O''_t whenever goal observations are not available, which is equivalent to a multiplication by a constant, and – when normalized – becomes effectively equivalent to a uniform.

- $P(S_t \mid S_{t-1})$. Finally, transition probabilities are stored A in the normal way for GHMMs, *ie* as the sum of the expected counts for the transitions.

6.3 Inference

We predict motion using the same two steps we have already explained for GHMMs, that we transcribe here for convenience. First, the belief state is reestimated:

$$P(S_t \mid O_t) = \frac{1}{Z}P(O_t \mid S_t)\sum_{S_{t-1}} P(S_t \mid S_{t-1})P(S_{t-1} \mid O_{t-1}) \tag{6.9}$$

[3] A block diagonal matrix is a square diagonal matrix in which the diagonal elements are square matrices of any size and the off-diagonal elements are zero.

it should be noted that, since goal observations are not available during prediction, observation probabilities are computed using exp. (6.2), which only takes into account the current state:

$$P(O_t \mid [S_t = i]) = \frac{1}{Z}\mathbf{G}([O'_t, 0];\ \mu_i,\ \Sigma) \tag{6.10}$$

Then, the extended state is propagated to the required time horizon H:

$$P(S_{t+H} \mid O_t) = \sum_{S_{t+H-1}} P(S_{t+H}\ S_{t+H-1}) P(S_{t+H-1} \mid O_t) \tag{6.11}$$

It is important to highlight the fact that the result of prediction is a probability distribution over discrete states. Sometimes, we are interested in knowing the probability of a particular point in continuous state space, for example, in the context of motion planning. This may be seen as the probability that a given state is observed in the future, and may be computed from the predicted state as follows:

$$P(O_{t+H} \mid O_t) = \frac{1}{Z} \sum_{S_{t+H}} P(S_{t+H} \mid O_t) P(O_{t+H} \mid S_{t+H}) \tag{6.12}$$

6.4 Structure and Parameter Learning

By introducing the intended goal into the state variable, we have obtained a state variable which is a good approximation to a *complete* state[4]. This makes it possible to apply algorithm 7 and perform parameter and structure learning in a straightforward fashion.

Thus, besides selecting the algorithm's parameters, the only additional task that should be performed is to preprocess trajectory data in order to include goal observations, before feeding it into the learning algorithm. For every input sequence of observations $O_{1:T} = \{O_1, \cdots, O_T\}$, the *augmented* observation sequence $\bar{O}_{1:T}$ is obtained by appending the last observation of the sequence to every observation vector $\bar{O}_{1:T} = \{[O_1, O_T], \cdots, [O_T, O_T]\}$.

This is similar to a supervised learning algorithm, where input data contains a label that identifies the motion pattern that has been executed by the object.

6.5 Learning example: a Unidimensional Environment

In this section, we will illustrate with an example the process of applying our GHMM based technique to a problem. Let use have the unidimensional environment depicted in fig. 6.2, where objects go from point A ($x = -5\,\mathrm{m}$) to point B ($x = 5\,\mathrm{m}$) or vice versa as indicated by the green and blue arrows. We assume that the probability of observing motion in one sense or the other is the same. Furthermore, we consider that objects move at $1\,\mathrm{m/s}$ and observations about the environment are sampled every second.

[4] A state is considered to be *complete* if it is the best predictor of the future , *ie* knowing the state, no prior variables may influence the evolution of future states [cf. Thrun et al., 2005]

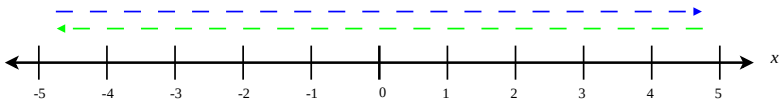


Figure 6.2: Example unidimensional environment. Objects move from A to B (blue) or from B to A (green).

6.5.1 Defining the state

It seems natural to start by defining the meaning of the state variable. A sensible choice is to assume that discrete states will represent the object's position. Thus, the extended state will be a two dimensional variable, meaning that for every discrete state i there will be an associated reference vector $w_i = [x_i, x_i']$, where x_i represents the object's current position and x_i' represents the intended position.

6.5.2 Choosing the parameters of the learning algorithm

From the fact that the velocity at which objects move in the environment is $1\,\text{m/s}$, and that observations are sampled every second, we know that objects may move $1\,\text{m}$ at every time step. Therefore, we decide that $1\,\text{m}$ is an adequate 'step size' for the current state, and we set the variance to the be the square of the step 'radius' $\Sigma' = [0.5^2]$. With respect to goals, we may be somewhat more flexible, and assume that two trajectories whose last states are within $2\,\text{m}$ of each other are indeed attaining the same goal, hence we set the intended state variance to be $\Sigma'' = [1^2]$.

For this particular case, we will set the insertion threshold to be $\tau = 1$, because our assumptions about the discretization size have already been encoded in the covariances. An alternative would be, for example, to increase the threshold and decrease the covariances. We do not want the initially found states to be smoothed, hence, we set the smoothing factor ε to zero.

We have now set all the parameters of our algorithm. It is very important to remark that we have not made any assumptions about the number of goals in the environment, the number of typical trajectories, or the size of the environments. Instead, we have used only our knowledge about the sensors (*ie* sampling rate), the objects dynamics (*ie* velocity) and the semantics of the state space (*ie* unidimensional positions). Therefore, these parameters may be applied to *any* unidimensional environment having the same kind of objects and sensors.

6.5.3 Learned model

Let us assume that, when objects go from A to B, the corresponding observation sequence is:

$$O_{1:11} = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\} \tag{6.13}$$

Of course, in most real situations, the observations will not be evenly spaced like these, because of the combined effect of noise, the sampling time, etc. But, we want to keep this example as simple as possible.

Now, we build the extended observation sequence, by appending the last observation in the sequence to every individual observation:

$$\bar{O}_{1:11} = \{[-5,5],[-4,5],[-3,5],[-2,5],[-1,5],[0,5],[1,5],[2,5],[3,5],[4,5],[5,5]\} \quad (6.14)$$

If we plot the points in the sequence, and draw the respective Voronoi regions and Delaunay edges, we will obtain something like fig. 6.3.

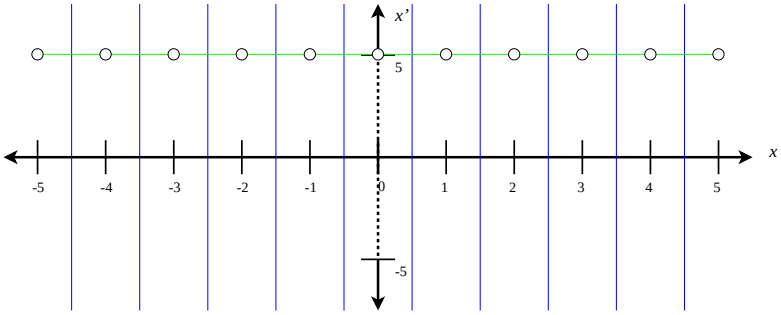


Figure 6.3: Extended observation sequence plot and Voronoi Diagram. Voronoi region's borders are depicted in blue and Delaunay links in green. x and x' axes are not at the same scale.

As we see, Delaunay edges[5] form a unidimensional manifold, which is a good representation of the structure of the input pattern. Our algorithm will add two transition probabilities for every Delaunay edge, and one self transition by node. So, after processing the extended observation sequence with the learning algorithm, we would obtain the GHMM illustrated in fig. 6.4.

In the figure, the transitions which have the strongest probabilities are those which advance the object towards B, while the probabilities of staying in the same state are lesser, and those of going backwards are close to zero. We may be tempted to remove those backward probabilities since, after all, they do not seem to encode any useful information, but we should remember that those edges were added before estimating the parameters, so it would not be possible to not include them without additional knowledge.

On the other hand, these low probability transitions only imply a linear increment in the complexity of the model and, due to their low probabilities they do not significantly affect the quality of prediction. Therefore, we considered that studying a mechanism to remove them is hardly justified.

Let us assume that now, we observe the trajectory of an object which goes from B to A, and the corresponding observation sequence is:

$$O_{1:11} = \{5,4,3,2,1,0,-1,-2,-3,-4,-5\} \quad (6.15)$$

[5] In this example, all Voronoi regions are infinite due to the fact that observations are all aligned over the same straight line – we say of this points that they are not in *general position* [cf. Guy, 1989] – hence, all borders are parallel. In normal circumstances, they would be slightly misaligned, so that the borders would cut each other thus defining finite regions.

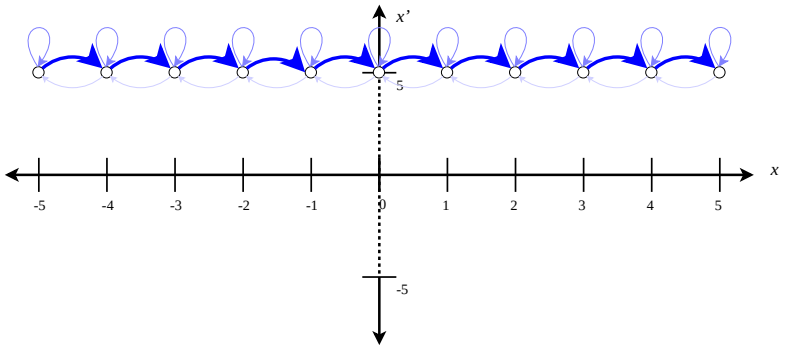


Figure 6.4: Learned GHMM after processing one kind of observation sequence. The size and color of the arrows represent the probability. x and x' axes are not at the same scale.

We may proceed as above, using our algorithm to update the model. At the end we will obtain the GHMM which is displayed in fig. 6.5.

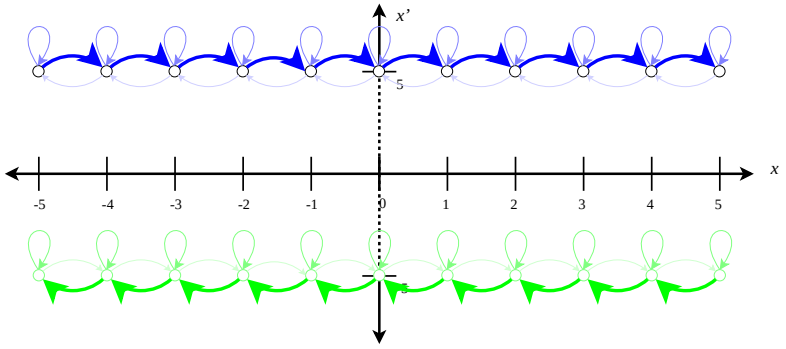


Figure 6.5: Learned GHMM after processing the two kinds of observation sequences. The size and color of the arrows represent the probability. x and x' axes are not at the same scale.

It is worth noting that, if we had allowed transitions for all the edges in the Delaunay triangulation, there would be edges joining both patterns. They are not there because our algorithm forbids the creation of links which are much longer than the insertion threshold.

As we have shown in this example, our algorithm has obtained a good representation of both motion patterns, without making any assumption about the number of typical trajectories or discrete states in the model. We will not discuss prediction because it is performed using exactly the same procedure than for conventional HMMs/GHMMs, except for the use of extended observations.

6.6 Comparison with existing HMM Based Approaches

One of the primary goals of this thesis was to develop a "learn and predict" approach, thus, the main difference between our approach and other HMM based motion prediction techniques is our incremental learning algorithm, however, this is not the only difference, we have also made a different set of hypotheses and modeling choices with respect to most approaches in the literature, which justifies a comparison in terms of this choices.

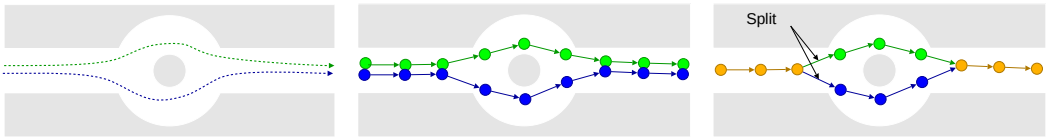
The two most important differences in our model with respect to other approaches are:

- We model motion patterns in terms of the intended state, while the 'standard' approach[6] is to consider that motion patterns correspond to "typical trajectories".
- While other approaches constrain the structure of a motion pattern to be a chain, our definition allows network-like structures, containing bifurcations and joining paths.

These two differences are relevant because they help to improve or solve on two problems related to the use of chains as typical trajectories.

Model Redundancy

A first problem that arises from the representation of motion in terms of typical trajectories comes from the fact that structures representing different motion patterns do not share states, which leads to redundant models. A simple example of this situation is illustrated in fig. 6.6. There are two similar typical trajectories (blue and green), which would be represented as two independent chains (fig. 6.6(b)) by applying the standard criterion. However, since both typical trajectories end in the same state and most of the intermediate positions are very similar, our algorithm would have produced something similar to fig. 6.6(c) which is a much more compact representation having the same expressivity.



(a) Typical Trajectories (b) Chains (c) GHMM

Figure 6.6: Shared state representation: Two typical trajectories (blue and green, left image) may be represented using two chain-like substructures (center). An alternative (left) is to use a single structure where the states shared by both structures (orange) appear just once.

An interesting situation arises when the occurrence probabilities for the different typical trajectories are not the same. Let us suppose that, in the above's example, the frequency with which the upper trajectory (green) is observed, is much higher than that of the lower one (blue). If we

[6] There are other goal oriented approaches in the literature (cf. § 3.5.2) but here we are only considering state space models.

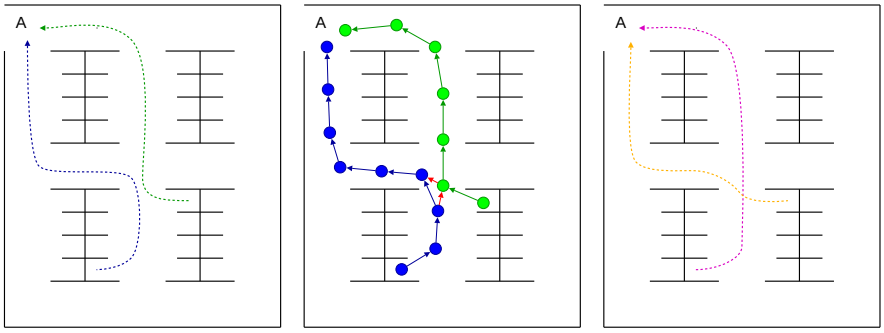
use chain-like structures, we may represent this by assigning a higher prior probability to the upper chain; unfortunately, most existing approaches assume an uniform prior, thus neglecting this information. By using our approach, the difference in frequencies would be encoded in the outgoing transition probabilities of the split point in fig. 6.6(c), where the object "chooses" to follow one path or the other.

Despite the advantages of sharing the state, it seems reasonable to ask if the fact of augmenting the state space dimensionality will not lead to an indiscriminate increase in the number of nodes in the Growing Hidden Markov Models (GHMM), thus affecting the applicability of this approach. The answer is no – at least when compared with typical trajectories approaches – because, in those approaches, two trajectories having different final states will, most likely, be represented as two different typical trajectories, thus leading to an increment in the number of nodes in the HMM which is, in the best case, equivalent to our approach.

A weakness that our algorithm shares with other approaches is its assumption that input trajectories have been correctly identified and segmented in a previous step. As we will see in chapters 7 and 8, this is not always the case, resulting in the addition of nodes for non-existent behaviors.

Pattern "Pieces"

When chains are used, transitions between different structures are not allowed, hence, the model is not able to explain motion patterns which are composed of "pieces" of existing patterns, thus ignoring valuable information which is already available. An example of this problem is depicted in fig. 6.7.



(a) Typical Trajectories (b) Chain-like (c) GHMM

Figure 6.7: Example of a situation where pattern "pieces" may be recombined to explain new behavior.

Model semantics

A deeper issue comes from the semantics of this "typical trajectories". They are always defined in terms of similarity, distance or other geometrical or statistical criteria without any reference

to the actual *causes* or factors that motivate motion. Therefore, these approaches do not address the crucial question "why are trajectories typical?" which – from our point of view – needs to be answered in order to produce truly generative models of motion. On the other hand, modeling motion causes is difficult in the case of people, because it implies taking the intentional stance towards moving objects. We consider that, although our approach is far from being a satisfying intentional model, it stands closer to a causal explanation of motion due to the fact that it is based on a rough model of intentions.

6.7 Discussion

In this chapter we presented the application of Growing Hidden Markov Models to learn and predict human motion. Our application is based on the hypothesis that motion is mainly determined by the object's intention to reach a particular state. We have explained how this intention is explicitly modeled by extending the definition of state in order to accommodate not only the current physical state of the object, but also the intended one.

We explained the particularities that should be taken into account to include the extended state into the GHMM formalism. We also provided a simple example of how a concrete problem may be solved using the proposed approach, and illustrated how no assumptions need to be made about the number of discrete states or the number of motion patterns to be learned.

We have compared our approach against other motion prediction techniques based on HMMs, explaining how our proposal leads to improvements in the compactness and the generality of the model.

Recapitulating, the technique we have presented:

- Is able to learn the structure and the parameters incrementally.
- Works on simple parameters which work on assumption about the dynamics of the objects and the capacities of the sensors, without requiring prior knowledge on the size of the environment or the number of motion patterns that objects execute in it.
- Produces models which are both rich and conceptually sound.

This concludes the exposition of the theoretic aspects of our work. The next part of this thesis is dedicated to the experiments we have performed to evaluate our approach.

Part IV

Experiments

Chapter 7

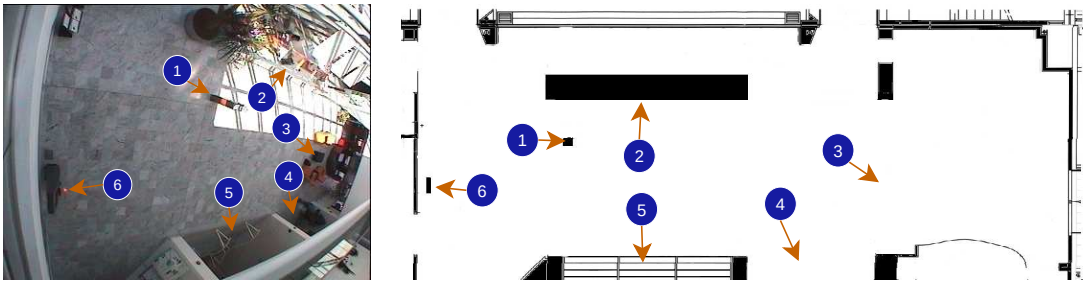
Experimental Platform

7.1 Overview

All of the experiments that we will present in this thesis are based on observation sequences which have been gathered on three different environments: the INRIA entry hall, the INRIA's parking, and a parking lot at Leeds University. One of the major difficulties we have faced during our research has been the acquisition of data coming from real sensors, hence, we have resorted to the use of synthetic data in order to complement our real data sets.

This chapter presents the three environments and discusses the obtention of the related data sets.

7.2 INRIA entry hall.



(a) video view (b) floor plan

Figure 7.1: Inria's main lobby and some landmarks:1) directory post, 2) front desk, 3) coffee area, 4) auditorium entrance, 5) entry stairs, 6) directory post.

The INRIA entry hall (fig. 7.1) features the main entrance to the building, two information directory posts, the front desk, a coffee area and a number of doors leading to various halls, rooms and auditoriums. This environment is the heart of the institute, all the personnel passes

through it at some point of the day for a reason or another (going in or out, coffee break, attending a lecture, etc.)

The most interesting feature of this environment is that it is – mostly – an open space. Since the environment's landmarks (*eg* front desk, doors) condition the way people moves, this environment may not be considered as completely unstructured. On the other hand, it is still very different from parking lots where the paths – at least for the vehicles – are made explicit by visual cues (*eg* floor markings) and physical obstacles (*eg* sidewalks).

On this environment we have obtained both real data coming from a tracker system, and synthetic data generated with a simulator. We will explain both cases in the rest of this section.

7.2.1 The visual tracker

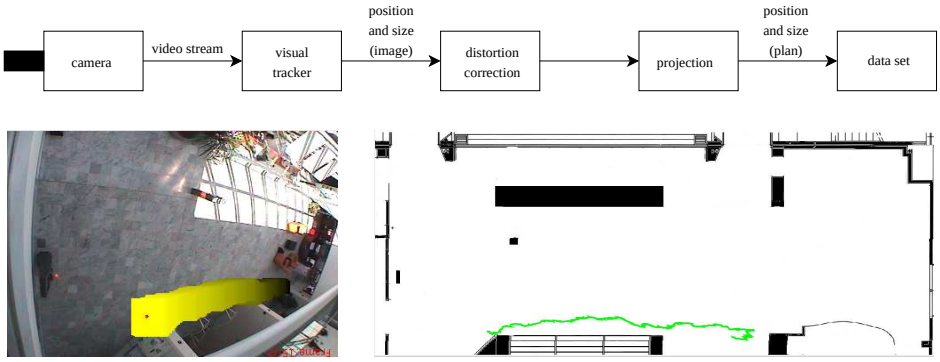


Figure 7.2: Architecture of the visual tracking system (top). An observation sequence presented in the image (bottom-left), and its projection in the floor plan of the environment (bottom-right).

To gather observations about the motion performed in the entry hall, we have used a visual tracking system for one camera mounted in a corner. The tracker has been developed by Gravir's team Prima and is described in Caporossi et al. [2004]. The system detects and tracks moving objects in a video stream coming from the camera. The collected information consists of the position and size (*ie* width and height) of the moving object in image's coordinates. This information is then projected into the floor plan of the environment. The data flow of the overall system is depicted in Fig. 7.2. It features the detector-tracker, a module to correct the distortion of the video camera, and a final module to project the information on the floor (*ie* the world plan). These modules are detailed in the upcoming paragraphs.

Camera and Tracker

A single wide-angle camera mounted over one of the lobby's corners is used. The camera is directly connected to the host computer. The tracker processes raw data coming from the camera and outputs data consisting of sets of observations, (*ie* frames), that the tracker sends at regular time steps. Every observation consists of an identification number (ID), the x and y coordinates of the moving object's gravity center in the image coordinate system, the width and height of the

object's bounding box and the orientation of this bounding box. Thus, a trajectory is represented as a sequence of such observations from track creation until termination.

Distortion correction and homographic projection

Due to the use of a wide-angle lens, the image is subject to heavy distortion, which must be corrected before projecting the image into the world coordinate system. We have used four-coefficient distortion correction as described in Zhang [2000].

A side-effect of the undistortion technique we have used is that a part of the image is clipped, thus reducing the portion of the environment which may be observed. On the other hand, the part that is lost corresponds to those places that are farther away from the camera (see fig. 7.3), where tracking performance is bad anyway.

(a) original (b) undistorted

Figure 7.3: Distortion correction on hall images. The effect is noticeable in the curvature of the left white line which appears in both images. Remark that a part of the original image is lost in the process

The corrected target gravity centers are multiplied by a precalculated homography matrix in order to project them into the world plane. It is worth noting that – most often – the target's center corresponds to a point which is located higher than the floor level, thus an error is introduced by projecting it into the floor. However, as the error is consistent for targets of similar height, we have decided that it is acceptable at this stage of our work.

Data Association

Due to the fact that we have an environment where there are multiple objects moving at the same time, the tracker does not always keep the ID of an object for all of its trajectory, which is a requirement for our approach. Hence, in order to improve ID keeping, we post-process projected data applying the Joint Probabilistic Data Association (JPDA) algorithm Bar-Shalom and Fortmann [1988] the fact of applying the algorithm on the world coordinate system helps to calibrate and improve the results of the tracking algorithm.

7.2.2 The simulator

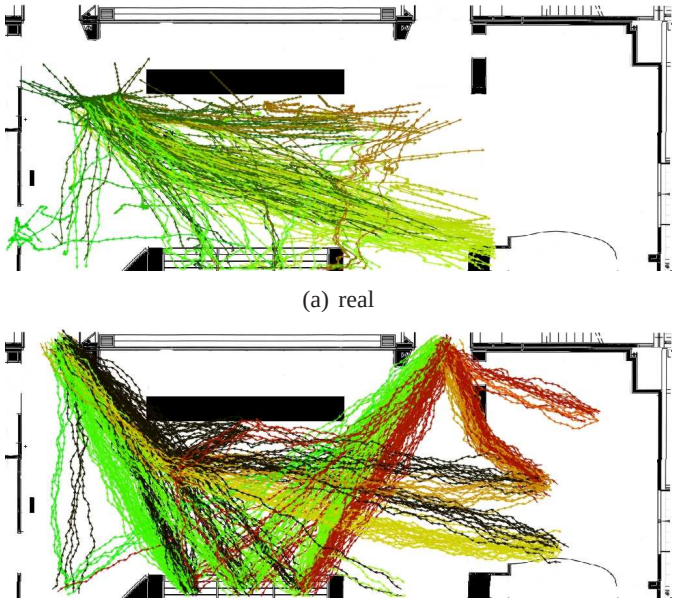
The simulator we have developed relies upon a number of control points representing "'places of interest" of the environment such as the doors, the front desk, etc.

Based upon this set of control points, a set of typical motions patterns has been defined. Each motion pattern consists of a sequence of control points to be traversed.

A trajectory is simulated in the following way: first, a motion pattern is randomly chosen. This motion pattern provides an ordered list of control points to which some Gaussian noise is added. Finally, motion between two control points in the list is simulated using discrete, even-size steps in in the direction of the next control point, corrupted with Gaussian noise. Switching from one control point to the next one is done when the distance to the current control point is below a predefined threshold. We have assumed a frame rate of $10Hz$ for our simulation. This process will be explained further when talking about Inria's Parking environment.

7.2.3 Data sets

We have collected real data during a week at different moments of the day, obtaining more than 3000 observation sequences. After filtering those sequences which where too short (less than 50 observations) or too long (more than 250) we kept a total of 2048 sequences, which are shown in fig. 7.4(a). For the sake of comparison, a set of 2000 synthetic sequences is presented in fig. 7.4(b).



(a) real

(b) synthetic

Figure 7.4: Hall data sets.

A first thing to notice is that trajectories in real data are sharply cut at the right and upper-

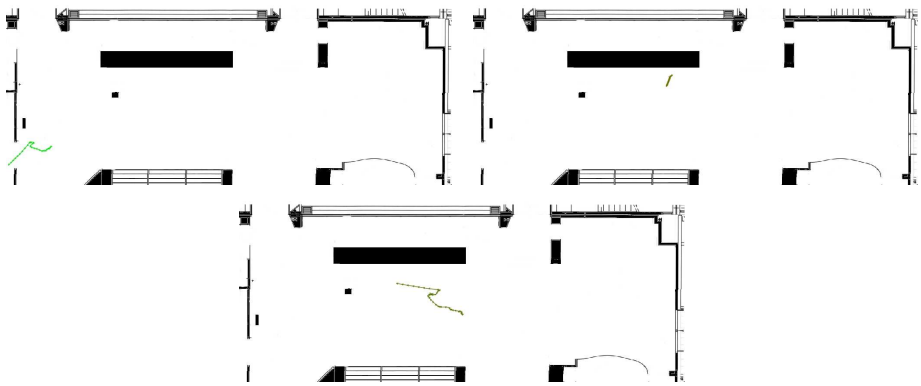


Figure 7.5: Anomalous trajectories.

right. This is a consequence of clipping during distortion correction (see §7.2.1). Conversely, synthetic data covers most of the floor plan.

As we might expect, synthetic data looks much more ordered and uniform than its real counterpart. However, this difference does not only come from the richer set of motions that humans may execute, but also from a number of sensor limitations. For example, some trajectories seem to pass through the front desk, which is probably due to tracking problems.

A problem which has more important consequences for our approach is posed by “anomalous" trajectories (fig. 7.5) which are due to tracking errors and do not correspond to real motion, or at least to complete motions. The problem is that these trajectories should not be used for learning, but they may not be automatically filtered because they are very hard to characterize. This seems to be a limitation of most current visual tracking systems, and most reported experiments on pattern based motion models use data sets which were corrected by hand [*eg* Hu et al., 2004b, Dee and Hogg, 2004]. In contrast, we have decided to keep data “as is”. We are aware that this will degrade the quality of our results for this data set, nevertheless we consider that it is worthwhile to test our approach against these conditions which, unfortunately, seem to be common to most current visual trackers.

A final difference between both data sets which is worth of notice is that there were some landmarks in the real environment that were not taken into account in the simulation. Two examples are the front desk and the door which is located near the bottom left corner of the floor plan.

7.3 Leeds parking data

The second experimental environment used in this thesis is a parking lot located at the University of Leeds[1]. This environment has at least two important differences when compared with the INRIA’s entry hall: first, it is populated by two types of moving objects (vehicles and pedestrians); and second, at least for vehicles, it is more structured than the hall environment.

[1]We would like to thank Hannah Dee and the University of Leeds for letting us use this data set.

The conditions on which data has been captured also differ from the hall environment. The video input has been captured by a camera located high above the parking, hence, a wide area is covered and the projective distortion is less pronounced than in the hall. As a matter of fact, we have decided to conduct our experiments in the image plane, thus eliminating the need for distortion correction and projection into the ground plane.

The tracking system used on this environment [Magee, 2004] is also different from the one already presented, but the main difference is that the trajectories have been hand-edited to correct problems like those we have discussed in the previous section. An indicator of the magnitude of this work is that approximately 20% of the 269 trajectories in the data set have been altered in some way, and some have been entirely tracked by hand [Dee, 2005].

Unfortunately, the reliability of this data set has a cost, and the number of trajectories in this data set is reduced. As a consequence, several behaviors are observed only once, hence, although they are learned, there are no more examples of the same motion pattern to evaluate the quality of the learned pattern.

The complete data set is depicted in fig. 7.6.

Figure 7.6: Leeds data set.

7.4 INRIA parking data

Due to the difficulty in acquiring data, we have decided – in addition to the real data sets that we have discussed above – to develop further our trajectory simulator (see §7.2.2) to dispose of a more thorough experimental testbed. This allows us to test particular situations without the logistic difficulties posed by executing scripted actions in a real environment as the INRIA's parking. It is also a practical way of generating big volumes of testing data which is free of the problems that are common to tracking systems.

In order to run a simulation, a graph-like structure should be defined (fig. 7.7). Nodes of the graph represent control points with an associated speed and position variance. They also have flags indicating whether they are the start or end points of a trajectory (or none). Nodes are connected by links, which indicate the possibility to go from one node to another.

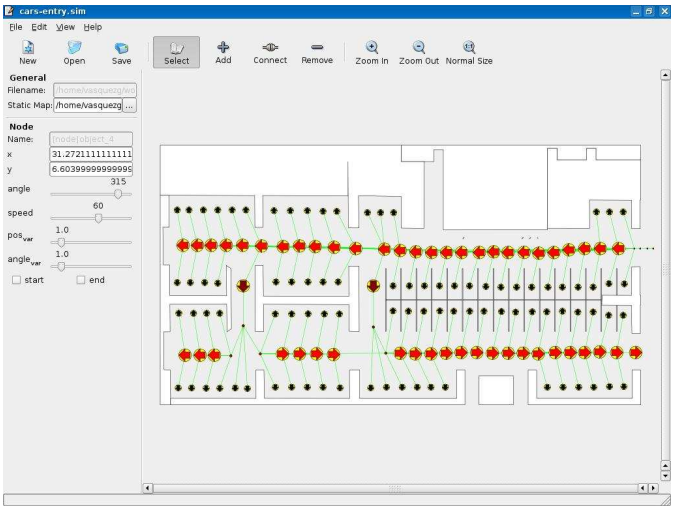


Figure 7.7: The trajectory simulator. Circled arrows represent control points: arrow size and color correspond to position variance and speed, respectively.

Once this graph has been defined, the simulator works by choosing at random a start and an endpoint (according to the corresponding flags), and then obtaining a sequence of nodes by finding the shortest path between the start and the end point. For every node in the sequence, a point is generated by adding Gaussian noise to the node's position. Finally, the trajectory is generated by applying spline interpolation to the point sequence and subsampling it according to the simulated sampling rate and object's speed. As for the entry hall simulations, we have assumed a frame rate of $10Hz$.

Fig. 7.8 shows an example trajectory data set obtained from the configuration shown in fig. 7.7.

7.5 Discussion

In this chapter we have presented the different environments and data acquisition platforms that we have used for our experimental work. For the case of real data, we have illustrated the difficulties that arise in the use of visual trackers as input systems for our approach and we have illustrated how some of these difficulties may be addressed. We have also explained our choice of using also synthetic data in our experiments.

The following table resumes the data acquisition platforms we have discussed in this chapter.

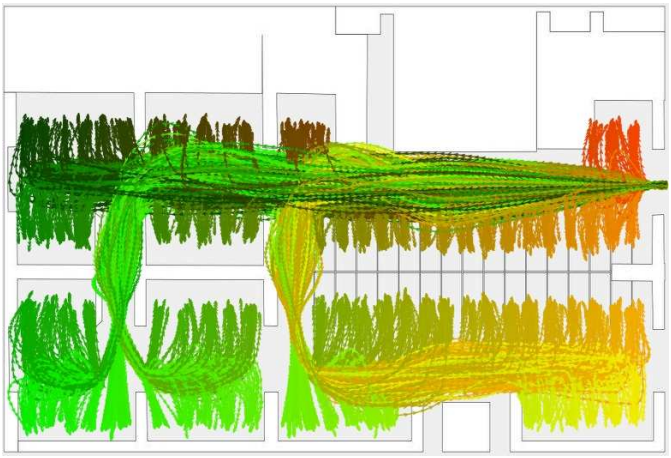


Figure 7.8: Synthetic parking data set, from the model displayed in fig. 7.7

Abbreviation	**Environment**	**Type**	**Observation**	**Object's speed**	**Trajectories**
IHR	INRIA's Hall	*real*	x,y	*variable*	2,048
IHS	INRIA's Hall	*synthetic*	x,y	*constant*	*as needed*
LP	Leeds' Parking	*real*	x,y	*variable*	269
IP	INRIA's Parking	*synthetic*	x,y,x',y'	*variable*	*as needed*

Every one of these combinations has interesting features which make them worth of analysis. We have two very different real data sets; one (IHR) consists on the raw output of the tracking system, covering a relatively small area, the other one (LP) has been manually cleaned, and covers a wide area, having two types of moving objects.

Synthetic data has also interesting differences. Hall data (IHS) comes from a small set of prototypes and all the objects are assumed to move at the same speed. Parking data, on the other hand, may correspond to very big number of prototypes and assumes that objects with variable speed. This last data set is also the only one to include richer observations, which include the object's speed. This makes it useful to test how our approach works on higher dimensional spaces.

As a concluding remark for this chapter, we will like to note that some tracking related problems have yet to be solved – or improved upon – in order to automatically produce the kind of data that our approach requires as input. In particular, we should stress the fact that robust data association – keeping a single identifier for an object while it is present in the environment – is indispensable for our approach to work.

Chapter 8

Experimental Results

> First you guess. Don't laugh, this is the most important step. Then you compute the consequences. Compare the consequences to experience. If it disagrees with experience, the guess is wrong. In that simple statement is the key to science. It doesn't matter how beautiful your guess is or how smart you are or what your name is. If it disagrees with experience, it's wrong.
>
> RICHARD FEYNMAN
> *From a PBS Show*

8.1 Overview

We have thoroughly tested our approach on the input data sets presented in chapter 7. Most of our experiments have been centered around the following key questions:

1. *Learning performance.* Does learning produce "appropriate" motion models? does it converge?
2. *Prediction accuracy.* How close are predicted states to real observed motion?
3. *Real-time applicability.* Are learning and prediction fast enough to work at camera rate?
4. *Generality.* Is our approach general enough for general motion prediction? (*eg* cyclic motion, higher-dimensional states).

The objective of this chapter is to present our experimental work on answering these questions. We will start by explaining some implementation issues, as well as discussing examples of the application of our approach to the different data sets. Then we will continue by providing numerical measurements of the behavior of our approach as related to our key questions, and close the chapter with a general discussion of our results.

8.2 Examples

In this section we present some examples of our approach at work in the different environments, without going into more technical details such as choosing the algorithm's parameters or data preprocessing.

8.2.1 The INRIA hall

Figure 8.2 shows a typical example of the prediction process using the IHR data set. It consists of a set of images arranged in columns and rows. Rows correspond to different values of t.

The left column shows the mean value of the predicted states for different time horizons going from $H = 1$ to $H = 15$, where H represents the number of time steps to look ahead in the future. These mean values are displayed as points colored from blue (for $H = 1$) to green (for $H = 15$).

The center column shows the probability of observing a particular state for $H = 15$, it has been computed by applying eq. (6.12) to the cells of a regular grid. Since the augmented state is 4-dimensional at least, we have chosen to project the probability over the current position plane, thus not showing the predicted goal.

The right column displays the current ($H = 0$) estimated goal. Similarly than for the center column, we have applied eq. (6.12) to the cells of a regular grid, but this time we have projected the probability over the intended position (goal) plane.

Images in the three columns share some elements (see fig. 8.1): a) the complete sequence of observations up to the current t, which is depicted as a series of red points; b) the current state and goal estimations, or expected values, which are pictured as red and yellow ellipses, representing the probability covariance.

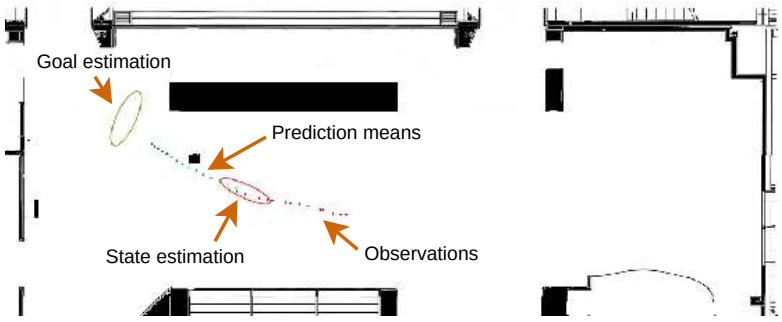


Figure 8.1: Common elements in prediction example images.

The images for $t = 1$ are interesting because of several reasons. When observing the environment, the person seems to "appear" in the middle of the hall, which is neither an entry/exit point nor a place where people stops for a while. This is a typical situation on this data set, and is due to the track initialization scheme used by the visual tracker.

Also noteworthy is the fact that, even having a single observation, the estimated probability for some goals (right image) is already high. At the same time, these goals are at opposed

sides of the environment, which seems reasonable: since only one observation is available and – for this data set – it does not include velocity, there is not enough information yet to infer the persons' direction. This is reflected in the roughly bimodal form of the predicted state (middle image). This also explains why the prediction means in the left image are centered on the persons' position.

In rows $t = 5$ and $t = 8$ it is possible to observe two parallel situations: the decrease of the estimated right goal probability and the emergence of a bimodal state prediction corresponding to the two left goals. At this moment, prediction means are also clearly orientated to the left.

At $t = 11$ the right goal has completely disappeared, and the same is starting to happen to the lower left goal. At $t = 15$ the final goal has been clearly identified, and the prediction means may be effectively regarded as the predicted trajectory which, in fact, is very close to the actual one followed by the object until it reaches its destination at $t = 44$. Indeed, in this case, the output of prediction algorithm closely resembles Kalman filter-based prediction once the goal has been identified.

8.2.2 Leeds data

Figure 8.3 show an example prediction sequence on the LP data set. Instead of describing the process in detail, we will focus on the difference with respect to the previous example.

A first contrast to notice is that predicted states at $H = 15$ seem to be considerably closer to the current object position. The reason is that, in this data set, objects move very slowly with respect to the size of the image.

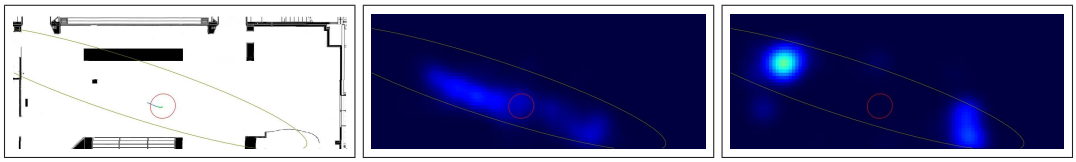
Another distinctness of this data set is that it includes two types of moving objects (*ie* pedestrians and vehicles). Since these objects follow different motion patterns, this has considerable influence in the prediction process. For example, for $t = 1$, we may see that there are two highly probable goals. This is interesting because they correspond to a pedestrian's destination (the building entrance) and a vehicle's destination (a lane's end). As the vehicle moves further, it becomes quickly associated with a vehicles' motion pattern and, by $t = 82$ the only two goals with a significant probability correspond to vehicles' destinations.

Also worth of notice is the fact that observations for this data set are expressed in camera coordinates, instead of referred to a floor plan. It is even possible to perceive the effect of the projective distortion: as the object gets closer to the camera, the gap between the predicted state and the object estimated position seems to increase.

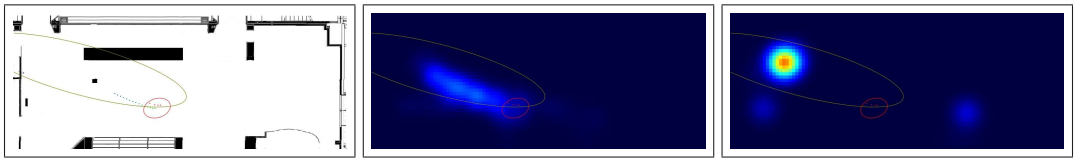
8.2.3 INRIA parking data

Our final prediction example is based on the IP data set which, besides being synthetic, has the distinctive feature of including velocities in the observations. The corresponding images are displayed in fig. 8.4.

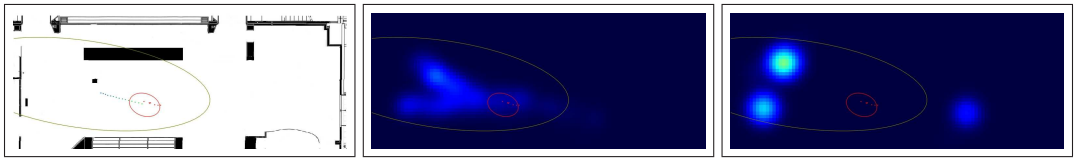
Something that makes this example stand apart from the others is the bigger number of goals in the environment. This is patent, for example, in row $t = 38$, where goals having a significant probability roughly cover half of the environment. Of course, this situation changes as the car continues to move and by $t = 49$, there are only two zones having a high probability of being the goal.



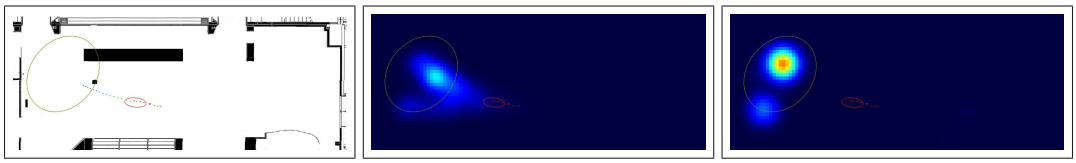
(a) t = 1



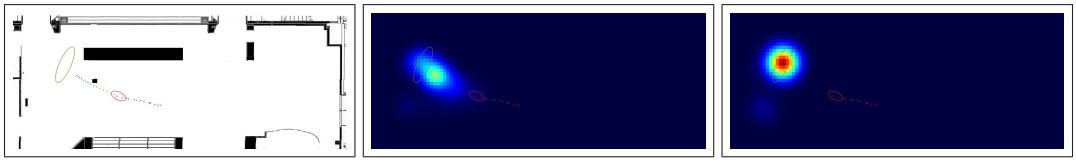
(b) t = 5



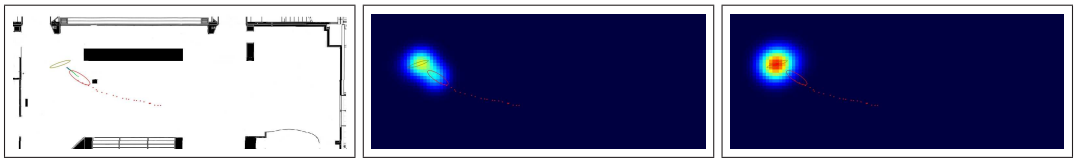
(c) t = 8



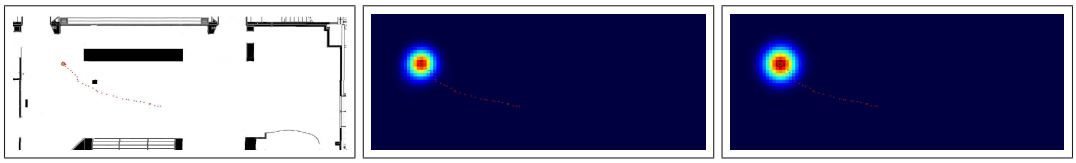
(d) t = 11



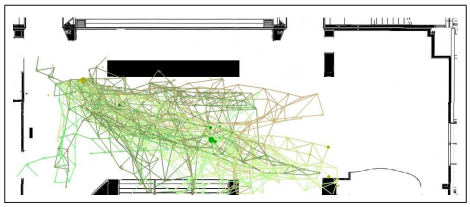
(e) t = 15



(f) t = 25

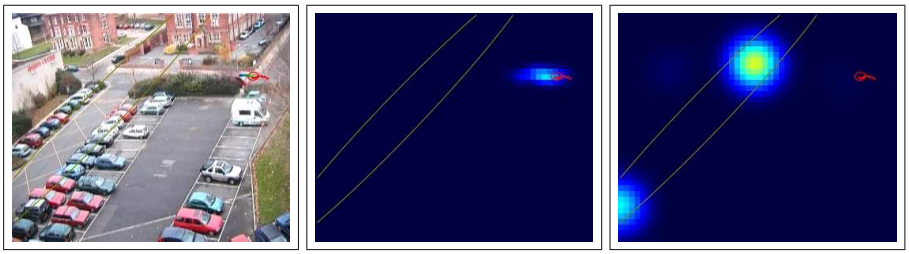


(g) t = 44

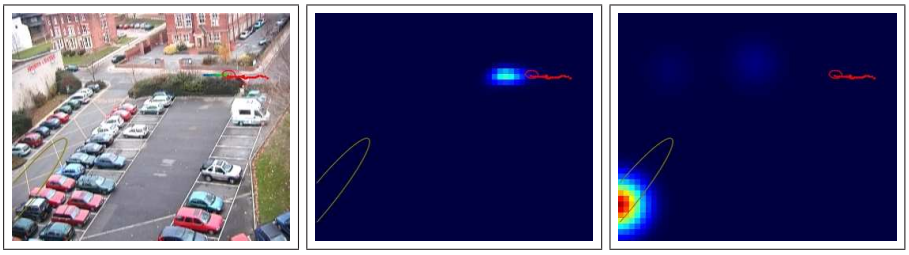


(h) structure

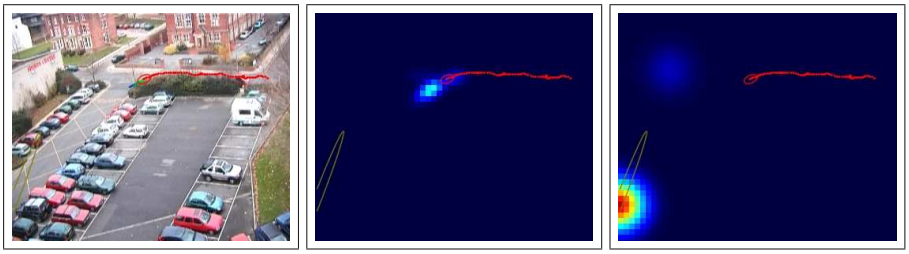
Figure 8.2: Prediction example. Hall environment–real data (IHR).



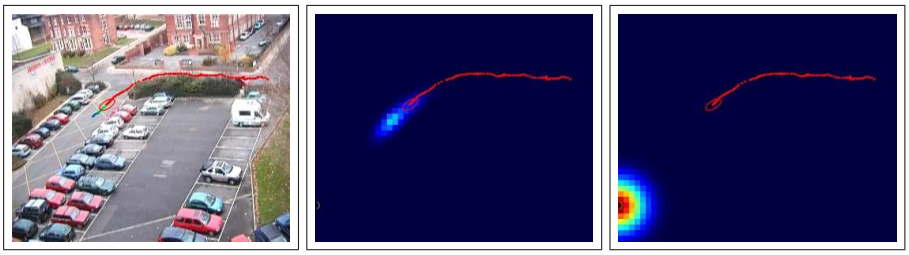
(a) t = 10



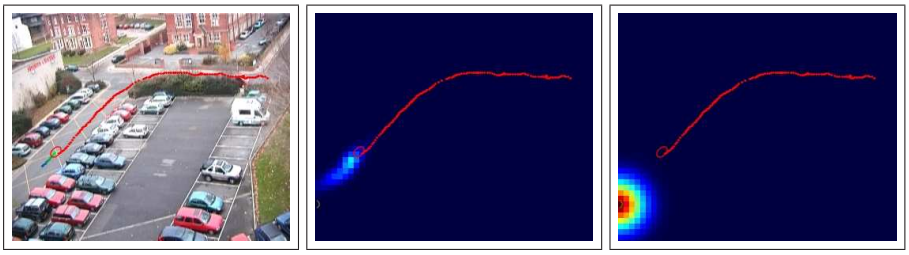
(b) t = 30



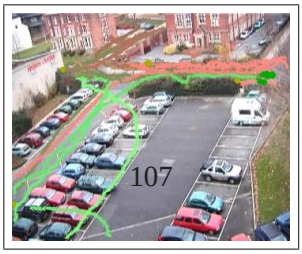
(c) t = 82



(d) t = 110

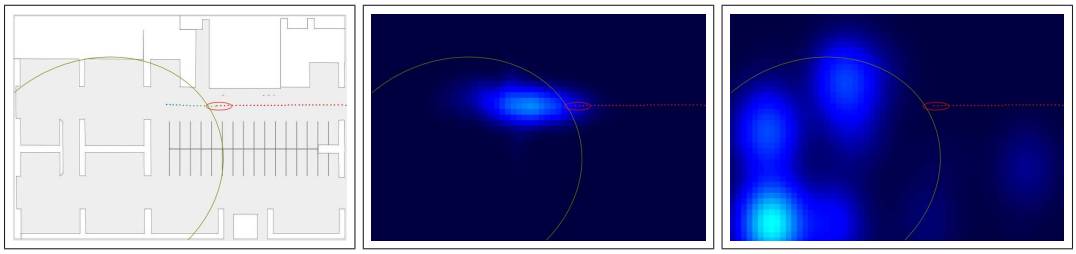


(e) t = 148

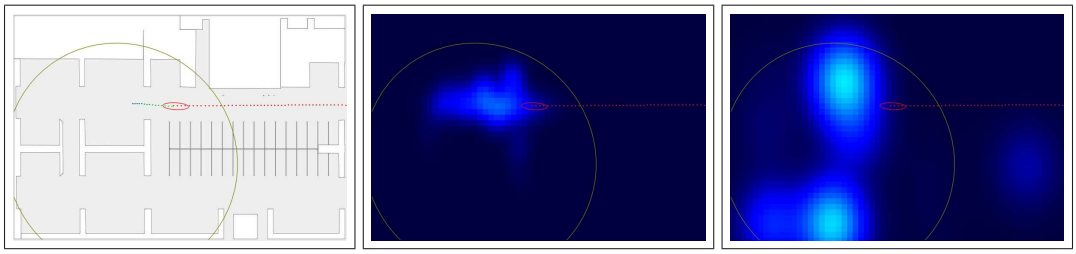


(f) structure

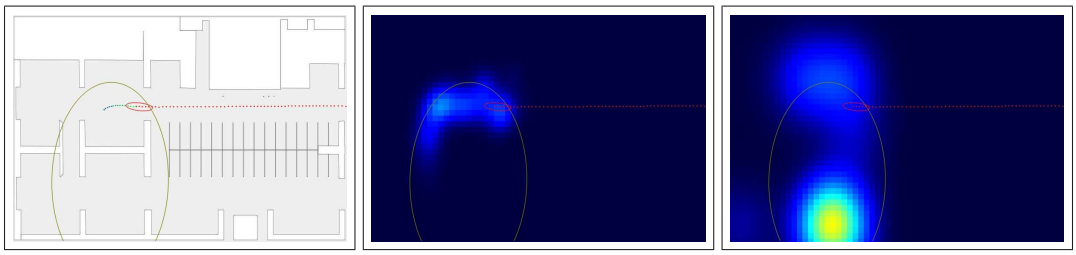
Figure 8.3: Prediction example (Leeds environment).



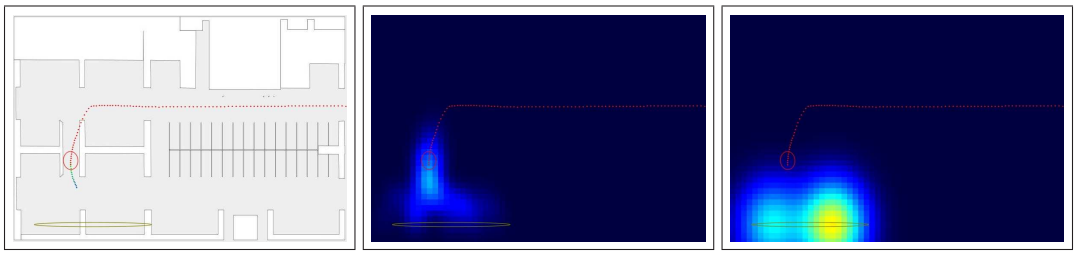
(a) t = 38



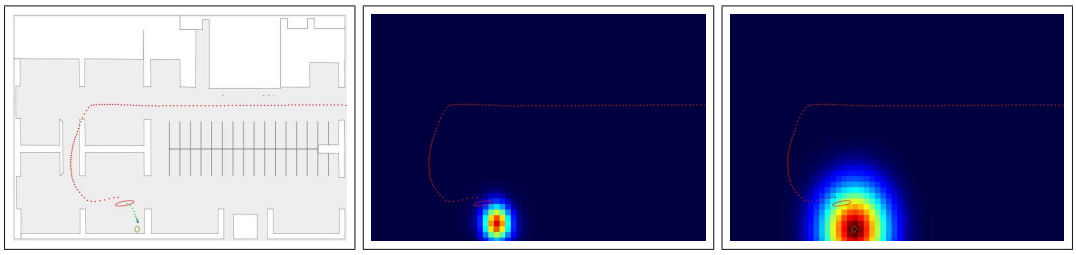
(b) t = 49



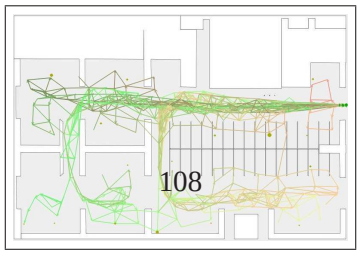
(c) t = 60



(d) t = 98



(e) t = 120



(f) structure

Figure 8.4: Prediction example (parking environment).

Having so many possible goals also influences the shape of the predicted state probability, which may become quite irregular, as for $t = 49$. This irregularity is also an indicator that a big number of alternative hypotheses that are being taken into account, which is unavoidable without additional information such as the free or occupied status of parking places.

Something that may not be appreciated in the images is that predictions are made in terms of the full state. This means that velocity information is also taken into account in order to increase the selectivity of the model.

8.3 Quantitative Results

In this section, we present the results of the experiments we have carried out in order to answer our key questions.

8.3.1 Parameter selection

For the purpose of studying the sensibility of our approach to parameter changes, we have used six sets of parameters in all our experiments, as described by the following table:

Data Set	**Units**	**Parameter Set**	**Threshold**	σ_{pos}	σ_{vel}	σ_{goal}
IHR	*decimeters*	Guess	4	15	–	25
		Low CV	4	12	–	20
		High CV	4	20	–	30
		Log IT	3	15	–	25
		High IT	6	15	–	25
		Best	9	12	–	20
IHS	*decimeters*	Guess	4	15	–	25
		Low CV	4	12	–	20
		High CV	4	20	–	30
		Low IT	3	15	–	25
		High IT	6	15	–	25
		Best	9	12	–	20
LP	*pixels*	Guess	4	10	–	20
		Low CV	4	7	–	10
		High CV	4	12	–	20
		Low IT	3	10	–	20
		High IT	6	10	–	20
		Best	9	7	–	20
IP	*meters*	Guess	4	2	0.04	5
		Low CV	4	1.5	0.04	3
		High CV	4	3	0.04	6
		Low IT	3	2	0.04	5
		High IT	6	2	0.04	5
		Best	9	1.5	0.04	4

In all cases, we have assumed spherical covariances for the state variables, hence the table only displays the corresponding standard deviation. In other words, for the IP data set, the covariance matrix has the form:

$$\Sigma = \begin{bmatrix} \sigma^2_{pos} & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{pos} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{vel} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{vel} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{goal} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma^2_{goal} \end{bmatrix}$$

And, for all other data sets:

$$\Sigma = \begin{bmatrix} \sigma^2_{pos} & 0 & 0 & 0 \\ 0 & \sigma^2_{pos} & 0 & 0 \\ 0 & 0 & \sigma^2_{goal} & 0 \\ 0 & 0 & 0 & \sigma^2_{goal} \end{bmatrix}$$

For every data set, parameters were fixed as follows: the first parameter set corresponds to an initial guess; the "Low CV" and "High CV" parameters correspond to smaller and bigger covariance values, respectively; the "Low IT" and "High IT" do the same thing for the insertion threshold; the last parameter set is the best one we have found with respect to its parsimony and prediction accuracy, in all cases it was obtained by trial-and-error over several combinations of insertion threshold and covariance values.

8.3.2 Measuring prediction accuracy

A common performance metric for probabilistic approaches is the maximum data likelihood or approximations like the BIC (see §4.6). However, for our particular application, this metric has the drawback of not having any geometric interpretation. Intuitively, we would like to know *how far* was the predicted state from the real one. Hence, we have preferred to measure the performance of our algorithm in terms of the average error, computed as the expected distance between the prediction for a time horizon H and the effective observation O_{t+H}.

$$\langle E \rangle = \sum_{i \in \mathcal{S}} P([S_{t+H} = i] \mid O_{1:t}) \| O_{t+H} - \mu_i \|^{1/2} \tag{8.1}$$

for a single time step. This measure may be generalized for a complete data set containing K observation sequences:

$$\langle E \rangle = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{T^k - H} \sum_{t=1}^{T^k - H} \sum_{i \in \mathcal{S}} P([S_{t+H} = i] \mid O^k_{1:t}) \| O^k_{t+H} - \mu_i \|^{1/2} \tag{8.2}$$

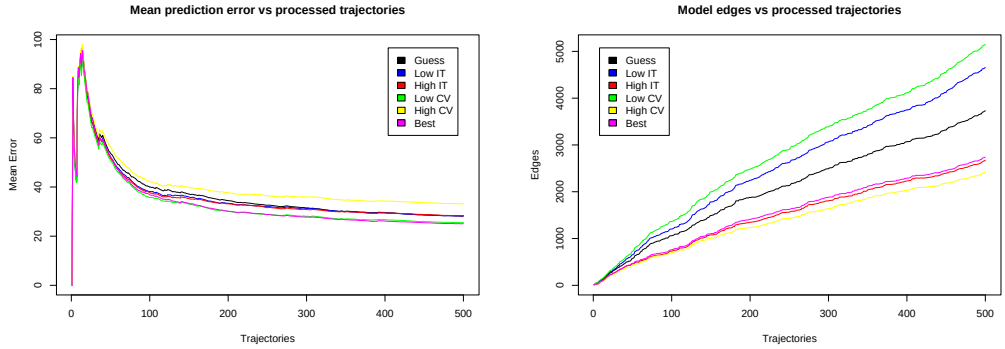
It is worth noting that, as opposed to the standard approach in machine learning of conducting tests using a "learning" and a "testing" data sets, the experiments we have presented here will use only a single data set. The reason is that, since learning takes place after prediction,

there is no need to such separation: every observation sequence is “unknown” when prediction takes place.

8.3.3 Hall real data (IHR)

For this data set we have performed some minimal preprocessing. In addition to the Joint Probabilistic Data Association (JPDA) filtering we have already discussed in §7.2.1, we have subsampled data by using only one out of three observations as input for our algorithm. The reason is that the position change that may be observed for pedestrians at full camera rate is far smaller than the detection noise: even assuming a relatively high mean speed of $5Km/h$, the position change between two consecutive frames at $24frames/s$ is about only $6cm$. Building a model with the required resolution would be very expensive with only marginal benefits, if any, due to the high noise/signal ratio.

Fig. 8.5 displays the behavior of our algorithm on a subset of 500 trajectories for the different parameter sets listed in §8.3.1. The left graph displays the evolution of the average mean error with respect to the number of trajectories that have been processed. The graph on the right shows the evolution of the number of edges in the model, as a measure of the model’s size.



(a) mean error

(b) model edges

Figure 8.5: Parameter sensibility (IHR data set)

A first thing to notice is that higher covariance values lead to a reduced number of edges at the cost of an increased error measure. This is a natural consequence of the fact that the “grain” of the discretization is in part determined by the covariance. Respectively, lower values for this parameter reduce the error measure, but increase the model’s size. It also appears that changes in the covariance are roughly proportional to changes in the number of edges.

However, the insertion threshold behaves less predictably, in particular in the case of higher values, which does not seem to affect the error measure considerably. This suggests that the model’s size may be decreased without sacrificing accuracy by using relatively high insertion thresholds. In fact, the best parameter set has been obtained by simply increasing the insertion threshold of the low covariance parameter set, obtaining a comparable performance while dividing the model size by two.

The rest of our analysis for this data set will focus on results obtained with the best parameters. Fig. 8.6(a) shows the evolution of both the model's size and the average error as a function of the number of processed trajectories. The first thing that may be noticed is that the model seems to be overfitting data. While the mean error starts to decrease very slowly after about 150 trajectories have been processed, the number of edges is still increasing at an almost constant rate. In other words, the model's growth has no important repercussions on prediction accuracy.

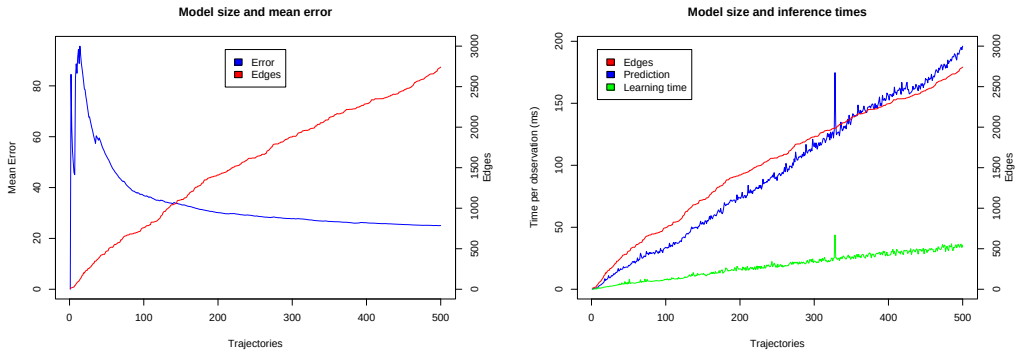


Figure 8.6: Error and computation times for the best parameters (IHR data set). Please take note that computation times are given per-observation.

After investigating the reasons for this effect, we have found out that the main cause is the trajectory splitting phenomenon already discussed in §7.2.3. Basically, observation sequences corresponding to single actual trajectories are being identified as independent, shorter trajectories. As a consequence, our algorithm identifies the ends of these smaller trajectories as goals and increasing the number of nodes and edges in the model accordingly. As we will see, this conclusion is supported by the results obtained with the other data sets, which are not subject to trajectory splitting.

In our opinion, this is one of the most significant problems faced by learning based motion prediction. Nevertheless, the importance of this problem has not been adequately highlighted by the existing literature. As a matter of fact, practically all the body of experimental work we have reviewed consists either of relatively small data sets obtained through more accurate (*eg* laser) sensors [*eg* Bennewitz et al., 2005] or has been preprocessed by humans [*eg* Dee, 2005, Hu et al., 2004b]. We will discuss this problem further at the end of this chapter and in chapter 9.

Fig. 8.6(b) shows a plot of the time taken by prediction and learning with respect to the number of trajectories that have been already processed, the number of edges is also plotted as a reference. Times are given per-observation, hence, in the case of learning, they should be multiplied by the length of the observation sequence, which for this data set was 50 on the average. As it may be expected, the increase in processing time has the same form than the model's growth. Since, as explained above, the model is too big, prediction is too slow for real time with about 0.2 seconds per observation in the worst case.

On the other hand, learning, in the worst case, took about 40 ms per observation, which

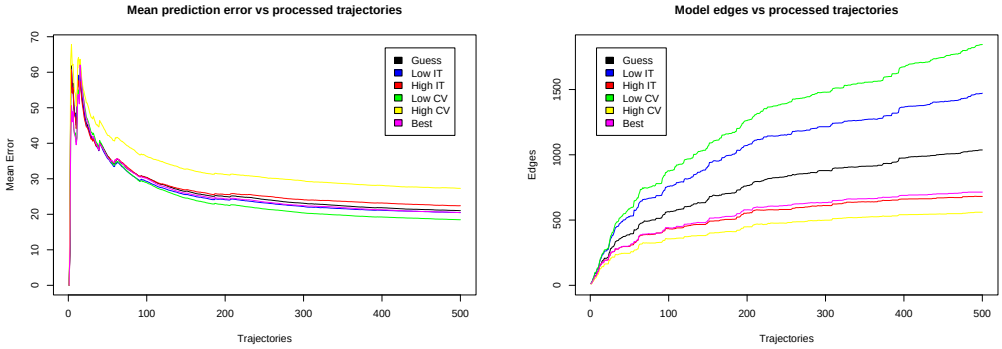
multiplied by the average 50 observations per trajectory gives a learning time of 2 seconds per trajectory, which we think is acceptable, given that learning takes place once the object has stopped moving.

Finally, we may observe that around the 330th trajectory, there is a sharp spike in processing times. This situation, which we will also observe in the other data sets, has been caused by some unrelated process which has been executed concurrently with our tests thus reducing available CPU time for our task.

8.3.4 Hall synthetic data (IHS)

In contrast with real hall data, this data set has not required a subsampling step since we have simulated a frame rate of 10 Hz. As for real data, we have processed a data set consisting of 500 trajectories, using the same set of parameters listed in §8.3.1.

Fig. 8.7 shows the parameter sensibility analysis for this data set where, as it may be expected, relative differences between parameter sets behave in a very similar manner to what has been described for real data. On the other hand, both the mean error and model size are considerably smaller.



(a) mean error (b) model edges

Figure 8.7: Parameter sensibility (IHS data set)

The differences with respect to the real data set become more evident when analyzing the evolution of the model's size and the average error (fig. 8.8(a)). The overfitting phenomenon observed on real data is not observable here, and the model size seems to converge together with the number of edges.

As for computation times (fig. 8.8(b)), we may observe that, again, the general form of the curve tends to follow that of the model size. This similarity notwithstanding, there is an important difference in the range of the curve. Now prediction time per observation is below 40 ms most of the time, with exceptions corresponding to reduced CPU time due to multitasking. This is suitable even for full camera frame rate operation at 24 Hz.

The same time improvement is observed in the case of learning times, with the same average trajectory length of 50 observations, now it takes only about 0.5 s to learn a new trajectory.

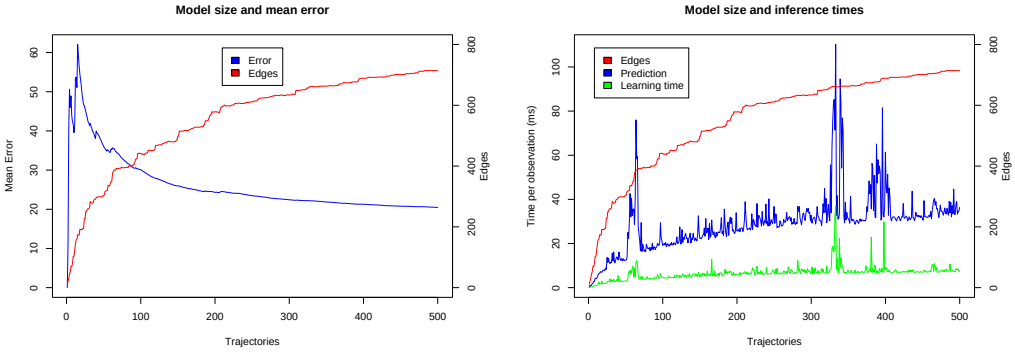


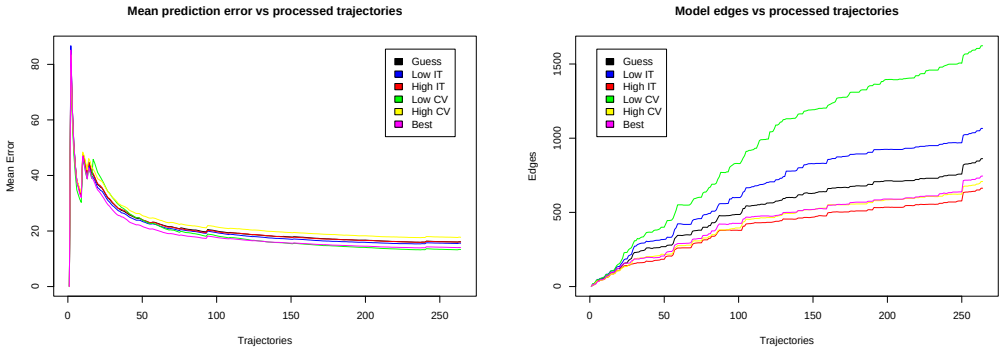
Figure 8.8: Error and computation times for the best parameters (IHS data set)

8.3.5 Leeds parking data (LP)

This data set has at least four features that make it very different from hall data: there are two different kinds of objects (vehicles and pedestrians) moving at very different speeds; prediction takes place on the image coordinate system; the number of available trajectories is limited; and, finally, there are some trajectories which correspond to motion patterns that have been observed just once. All these factors lead are reflected in the corresponding experimental results.

For this data set we have again performed a one out of three subsampling, for the same reasons already explained for data set IHR. However, if only cars were presented on the scene, it would be possible to go without subsampling, since cars move much faster than pedestrians.

As shown in fig. 8.9, parameter sensibility works similarly to the other experiments even if "Low COV" parameters seem to produce a slightly more important model size increase than in previous cases.



(a) mean error (b) model edges

Figure 8.9: Parameter sensibility (LP data set)

The growth in model size and the error evolution with respect to processed trajectories (fig. 8.10(a)) were distinctive. There seems to be an overall convergence process combined with sudden stair-like increases in both model size and average error (see for example the graph at 100, 160 and 250 processed trajectories). Actually, these increases correspond to the unique motion patterns that we have mentioned above.

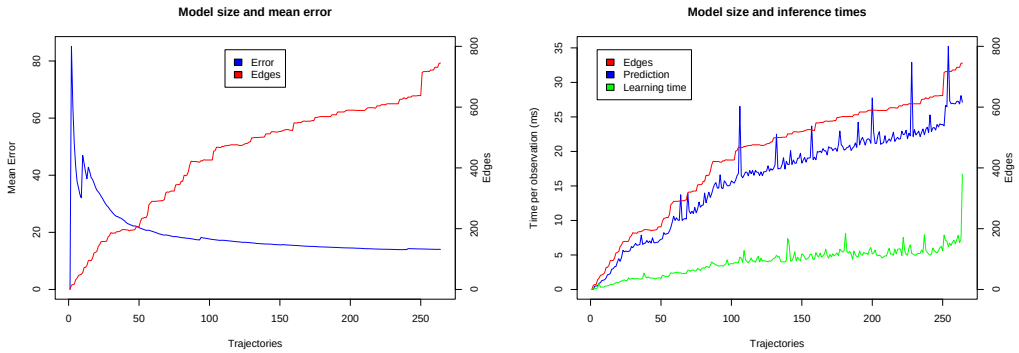


Figure 8.10: Error and computation times for the best parameters (LP data set)

As for previous cases, learning and prediction processing times increase according to the growth in model size. Moreover, even in the worst case, prediction does not take more than 25 ms per observation, which is almost the double than normal camera frame rate. As for learning, it has always been less than 1 s per trajectory.

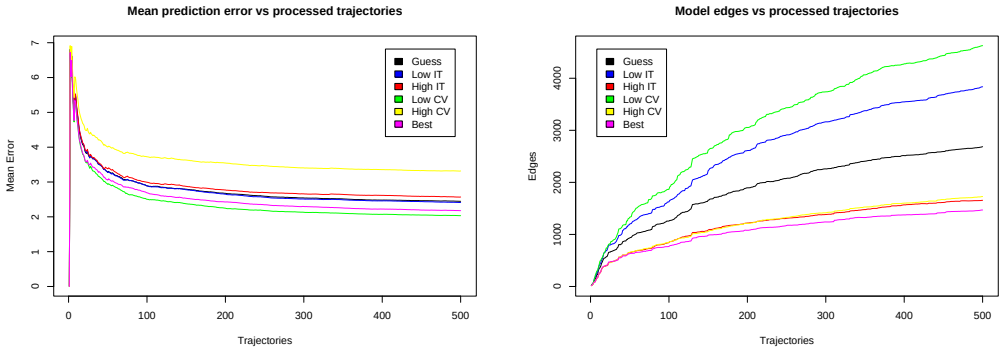
8.3.6 Inria parking data (IP)

This last data set presents has two distinctive features: observations include velocities, meaning that the learnt structure occupies a six-dimensional manifold; and a big set of possible destinations, with about 90 parking places. As we will see, our algorithm performs surprisingly well, taking into account this added complexity.

In the parameter sensibility graphs in fig. 8.11 we see that, in general, the algorithm behaves as normal, but for this data set the best parameter set has produced the smallest model – by a small margin – while having the second best average error.

Given the size of the data set and its features, it was surprising to find out that the error evolution and growth (fig. 8.12(a)) in model size performed that well, with a final number of edges below 1500 despite the big number of goals (compare with the IHS). We believe that the main reason for this reduced size is that, due to the environment's structure, trajectories leading to goals in the same area tend to share a considerable number of states.

Because of the moderated model size, time performance was correct, with a prediction time of little less than 60 ms, and an average learning time of about 3 seconds after 500 processed trajectories. Even if prediction times are slightly slower than camera frame rate, we think that these are very good results, taking into account the characteristics of this data set.



(a) mean error (b) model edges

Figure 8.11: Parameter sensibility (IP dataset)

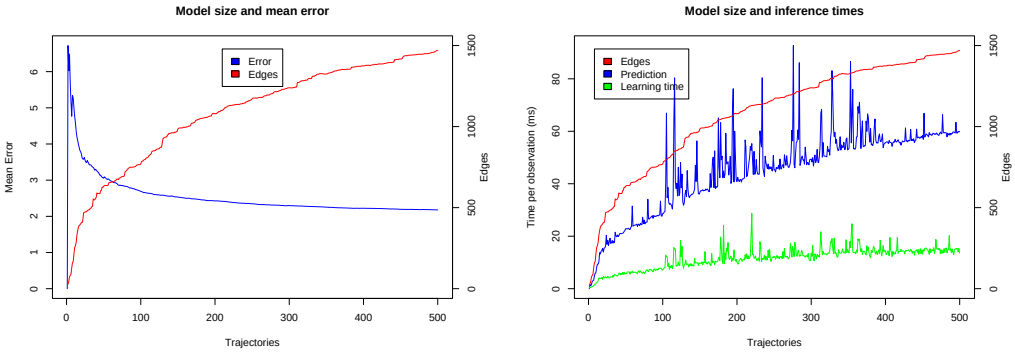


Figure 8.12: Error and computation times for the best parameters (IP data set)

8.4 Modeling motion with cycles

A question that has been often posed to us is how our approach deals with motion patterns containing cycles. As an example, let us imagine an 800 m track running event, where runners have to go two times around the circuit in order to finish the race. It is evident that the situations at 395 and 795 meters are very different, in the first one, the competitor will continue to run for another turn, while in the second one it will stop after five meters. Since our motion prediction approach as it is defined, only distinguishes discrete states in terms of their position and speed, it is not able to deal with this situation, resulting in equiprobable predictions of stopping and continuing at both 395 and 795 meters.

Although this example may seem contrived, this situation arises frequently for other kinds of motion, for example for gesture recognition. Here, we propose a simple solution to this problem: including the time variable t in observations. To illustrate this, we have prepared a synthetic data set consisting of two symmetric motion patterns presenting several nested cycles (see fig. 8.13).

The only difference between both of them is that one always turns to the left and the other one to the right.

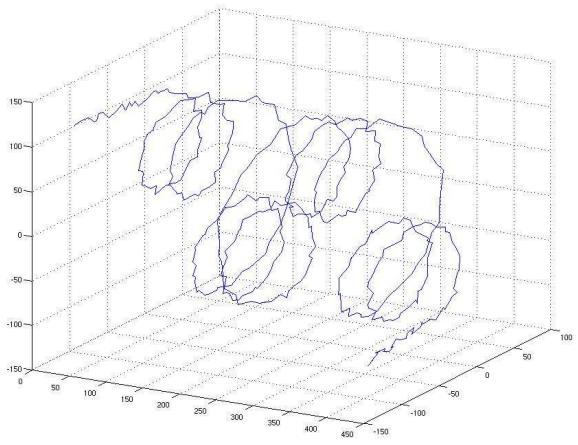


Figure 8.13: Example of cyclic trajectory. The time axis goes from 0 to 450. Notice how it always turns to the right, the other pattern is similar, but it only has left turns.

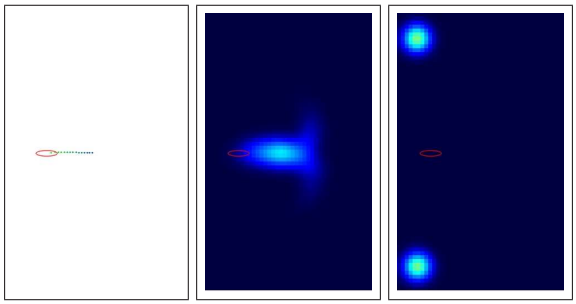
Of course, including the *time* variable on observations implies augmenting the observation covariance matrix, for our example it has the form:

$$\Sigma = \begin{bmatrix} \sigma^2_{pos} & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{pos} & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{goal} & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{goal} & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{t} \end{bmatrix}$$

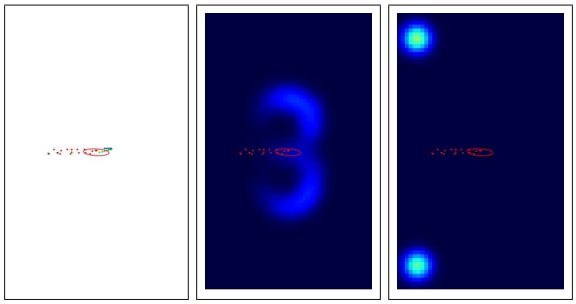
Without going into the parameter details, fig. 8.14 illustrates our algorithm working in this data set, here the structure for the two motion patterns may be clearly seen at the bottom, where gray corresponds to left turns and green to right ones.

Notice how object's positions for $t = 20$, $t = 60$ and $t = 100$ are very similar. Nevertheless predictions are clearly different. At $t = 20$ the object has not even turned, hence, the estimated probability for both goals practically the same. After one turn, at $t = 20$, the algorithm has identified the left turning motion pattern, and it predicts that the object will still perform one more small turn. Finally, at $t = 100$ the algorithm recognizes that the object has performed two small turns and thus predicts that the object will continue to follow the big one.

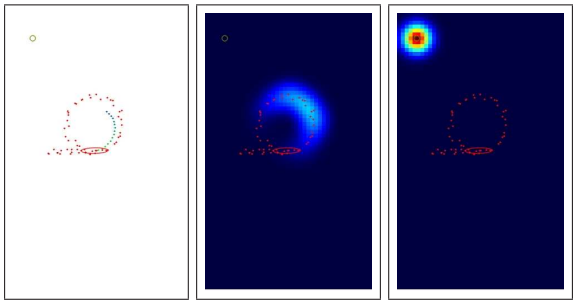
As our example shows, by simply redefining observations, our algorithm is able to deal with cyclic motion patterns without further modifications. Moreover, moderated temporal misalignments may be easily dealt with by increasing the value of σ_t .



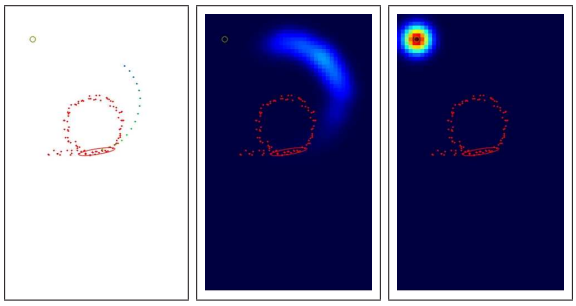
(a) t = 1



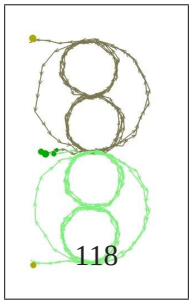
(b) t = 20



(c) t = 60



(d) t = 100



(e) structure

Figure 8.14: Prediction example for the cycle data set.

8.5 Discussion

In this chapter, we presented the results of applying our learn and predict approach to the data sets introduced in chapter 7, with the intention of studying four points: a) the performance of the learning algorithm; b) the accuracy of the obtained predictions; c) the real time applicability of the whole approach; and d) the generality of our approach and its applicability to data sets having higher dimensions, or containing cyclic motion.

We illustrated with examples how our algorithm behaves on the different environments, illustrating along the way some of the problems which arose in the implementation. We explained the need for a performance measure, and proposed the average prediction error to evaluate our algorithm. Finally, we presented the quantitative results we have obtained using different sets of parameters for every data set.

The somewhat bad results that we have obtained with the "raw" IHR data set, highlight an important problem of our approach, which is shared by similar techniques in the literature: the assumption of having a reliable sensor layer, which is able to robustly solve the data association problem and, thus, output complete observations sequences. Unfortunately – at least in the case of visual trackers – this condition is not met by current systems, making it necessary to investigate ways to relax this assumption, which, from our point of view, is a primordial direction for future work.

On the other hand, we believe that the results we have obtained with the other data sets show that, when complete trajectories are available, our algorithm is able to learn motion patterns and produce accurate predictions in real or near real-time, even for higher dimensional state spaces and for cyclic behaviors.

Part V

Conclusion

Chapter 9

Conclusions and Future Work

9.1 Conclusions

In this thesis we focused on the problem of predicting motion of mobile objects which are able to modify their trajectories at will, as is the case for persons, vehicles and the like. Our work builds upon a family of approaches which are based on the idea that, for a given environment, objects follow typical motion patterns which may be observed consistently.

As discussed in ch. 3, most of these approaches operate in a *learn then predict* fashion [*eg* Walter et al., 1999, Makris and Ellis, 2002]: first, they learn motion patterns from sensor data, then, after learning has been performed, they predict further motion on the basis of learned patterns. As we have shown, this has a number of drawbacks, the most important being the inability to improve acquired knowledge when a previously unknown motion pattern is observed.

In contrast, we have aimed to find an approach with the capability of continuously refine its knowledge using the same observations that constitute the input for prediction. We have decided to base our work on Hidden Markov Model, a probabilistic framework, which is very popular in the literature [*eg* Koller-Meier and Van Gool, 2001, Bennewitz et al., 2005]. However, unlike these approaches – which rely on clustering algorithms to identify motion patterns – we have set to integrate the whole learning process into an HMM parameter and structure learning algorithm.

Our approach if based on the idea that a single HMM may represent multiple motion patterns by defining an appropriate graph structure. This idea is already present in other approaches, where different motion patterns correspond to different non-connected components of the HMM structure. Therefore, learning motion patterns may be seen as an HMM structure learning problem, with the additional constraint of working incrementally, due to our learn and predict requirements.

Unfortunately, as we have shown in ch. 4, no structure learning algorithm in the literature seems to be fit to our problem. This has led us to develop a novel incremental HMM structure and parameter learning algorithm.

The main difficulty behind incremental structure learning on HMMs comes from the huge size of the space of possible structures, which renders search-based approaches unpractical for our purposes. Hence, we have taken a different approach, since, in our case, an HMM is just a discrete approximation of a continuous phenomenon, it is possible to impose constraints to the

HMM structure according to the topological properties of continuous state evolution.

Given that HMM states correspond to discrete regions on the state space, it follows that continuous motion may only progress by traversing neighboring regions, which significantly limits the number of possible transitions between states. This observation has suggested the use of a topology learning network known as the Instantaneous Topological Map to learn the HMMs topology. For this network, learning is both incremental and adaptive, hence, our structure learning algorithm also has this properties.

Of course, a complete HMM learning algorithm should also estimate the model's parameters, thus, we have interleaved our topology learning algorithm with an incremental HMM parameter learning algorithm based on the one proposed by Neal and Hinton [1998]. This results in a general procedure for learning the structure and parameters of an HMM, which is presented in ch. 5. In contrast with traditional HMMs, the number of discrete states on the model changes with time – mostly growing – thus the name "Growing Hidden Markov Models".

As we show, GHMMs, are well suited for the common case in which the model is used as an approximate inference tool for continuous state spaces. However, in order to apply them successfully to the motion prediction problem, further assumptions should be made. We have explained the problem in 6 and proposed a solution: integrating the final state of the object into the state vector. Since this information is available during learning this may be regarded as a semi-supervised learning approach, in which the final destination or goal constitutes a class label.

We have implemented our approach and demonstrated with experiments on real and synthetic data (chapters 7 and 8) that our approach:

- Is able to learn motion patterns and represent them by incrementally adapting the structure of the GHMM.
- Is able to perform state estimation, as well as prediction for an arbitrary time horizon, using a probabilistic belief representation.
- Is able to perform early identification of an object's destination.
- It is able to perform both learning and prediction in real time even on complex environments and situations.
- Is defined in terms of intuitive parameters which do not require prior knowledge on the form of the environment or the number of motion patterns to be learned.

We consider that these results are encouraging, however, we are aware that pattern based motion prediction techniques are still at a very early state. In consequence, during our experiments and – in general – our research, we have found several areas for improvement, which will be discussed on the next section.

9.2 Future work and possible extensions

We see two general directions to extend the work presented in this thesis. The first one consists in extending the approach at a high-level of abstraction, specifically by taking into account

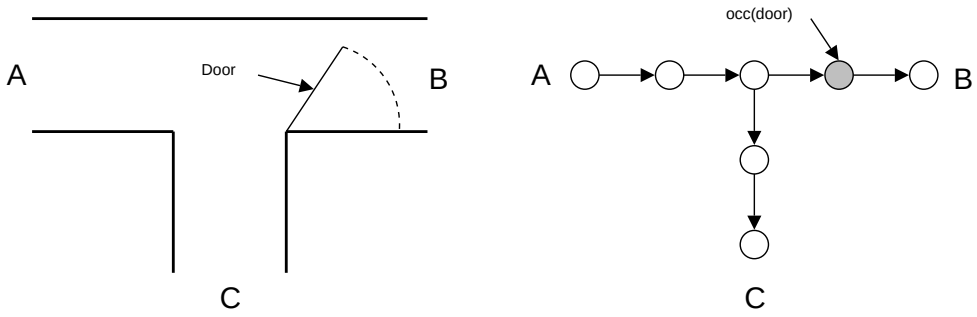
the possible interactions between objects. The second one consists in improving the low-level, making the approach more robust to the limitations of current video trackers and sensors in general.

9.2.1 High-level extensions

Currently, our approach does not take into account the possibility that an objects modifies its trajectory in response to the motion of another object. Here, we will discuss how our approach may be extended to handle some specific situations of this type.

A situation which is relatively simple to represent in our approach is the case where there are semi-static obstacles in the environment, *ie* objects which are not able to move freely, but may be in one of two possible states (*eg* 'block' or 'not block') which condition the choice of paths that other object may follow. This is the case, for example, of doors, which may be either 'open' or 'closed', or parking places, which may be either 'free' or 'occupied'.

Assuming that the information on the state of these obstacles is available via a sensor, extending inference is relatively simple. The key idea is that, when an object is is in a 'block' state, it hinders other object's access to certain states. We will illustrate this idea on the T-shaped environment presented in fig. 9.1(a).



(a) Environment (b) HMM structure

Figure 9.1: Example of an environment with a semi-static obstacle (door). The structure assumes that motion starts from point "A"

A simple HMM structure for this environment is presented in fig. 9.1(b). When the door is closed, it "occupies" the grayed-out state. Although in our example there is only one such state, in general we will use the notation $occ(obstacle)$ to denote the set of states that a static object occupies when it is in its block state.

In order to perform inference, we augment our model with a variable $S_{door} \in \{open, closed\}$, indicating the state of the door, modifying the JPD as follows:

$$P(S_t\ S_{t-1}\ O_t\ S_{door}) = P(S_{t-1}\ S_{door})P(S_t \mid S_{t-1}\ S_{door})P(O_t \mid S_t\ S_{door}) \tag{9.1}$$

Since, when the door is closed, no object can be at the $occ(door)$, it follows that:

$$P(S_{t-1}\ S_{door}) = \begin{cases} 0 & \text{if } S_{t-1} \in occ(door), S_{door} = closed \\ P(S_{t-1}) & \text{otherwise} \end{cases} \tag{9.2}$$

For analogous reasons, transitions to an occupied state are forbidden:

$$P(S_t \mid S_{t-1}\ S_{door}) = \begin{cases} 0 & \text{if } S_t \in occ(door), S_{door} = closed \\ P(S_t \mid S_{t-1}) & \text{otherwise} \end{cases} \tag{9.3}$$

and there is a null probability that an observation comes from an occupied state:

$$P(O_t \mid S_t\ S_{door}) = \begin{cases} 0 & \text{if } S_t \in occ(door), S_{door} = closed \\ P(O_t \mid S_t) & \text{otherwise} \end{cases} \tag{9.4}$$

These definitions may be easily extended for multiple semi-static objects, and used to perform inference using standard methods. Information about semi-static objects may be supplied by an expert, but fully unsupervised learning becomes considerably harder: not only it is necessary to identify semi-static objects, but also the states their block.

A much more challenging extension, would be taking into account interactions between fully dynamic objects. Normally, this would imply modeling the joint state of all the objects that are present in the environment at the same time, but this is clearly intractable. Instead, we believe that it would be interesting to investigate how to integrate factored solutions, like CHMM [Brand et al., 1997] or DML-HMM [Gong and Xiang, 2003, Xiang and Gong, 2006].

Another alternative would be to integrate our approach with behavioral models like the one proposed by Oliver et al. [2000], which produce high level inferences like “object x is approaching object y", but are unable to produce metric information which is useful for motion planning. It would be very interesting to adapt our structure learning to provide a ’metric layer’ for this kind of approaches.

9.2.2 Low-level extensions

As most pattern based approaches, our approach is based on the assumption that input is available in the form of complete observation sequences for every object. Nevertheless, as discussed in chapter 7, this assumption does not hold in practice, at least when using visual trackers. As a consequence, many motion patterns which do not always correspond to the actual object behavior are learned, unnecessarily increasing the model’s size and significantly affecting prediction accuracy. This problem has been addressed in the literature by implementing “anomaly detection” algorithms [*eg* Hu et al., 2006] to filter out abnormal trajectories, unfortunately, they rely on having complete data sets and are designed to work in batch mode.

We believe that it would be better to explore ways to increase the robustness of pattern based approaches by relaxing this “complete trajectory" assumption. We think that, in order to achieve this, it will be necessary to get closer to the tracking system. A possibility is to study means to integrate our approach into the prediction step of the tracker.

Indeed, some steps have been already taken in this direction. Two noteworthy examples are: [Liao et al., 2003] and Bennewitz et al. [2005]. However, there is still a lot of work to do: the first approach assumes that a map of the environment is known *a priori*, while the second one

assumes that the learning if performed on “complete trajectories” which is a circular problem, unless a more reliable sensor (*eg* a laser scanner) is used for learning.

Appendix A

Notation and Abbreviations

Here, we present a summary of the notation and abbreviations that we have used throughout this document. We have also adopted the standard convention of denoting random variables as capital letters, and values as lower-case letters. For logical propositions, we have used capital calligraphic characters, or expressions enclosed between brackets. When defining parametric forms, or working with linear algebra, vectors are represented as lower case letters to distinguish them from matrices, which are always capitalized.

List of Symbols

$C(\mathcal{A})$	The number (count) of cases in which the proposition $\mathcal{A}$ is true in experimental data.
D_V	The domain of V.
$\mathbf{G}(V;\ \mu,\ \Sigma)$	The gaussian distribution on V, with mean μ and covariance Σ.
O_t	The observation corresponding to time t.
S_t	The state of the object/system at time s.
t	The current discrete time step.
$\mathbf{U}_V$	The uniform probability distribution on V's domain.

List of Acronyms

AHMM..... Abstract Hidden Markov Models *page 33*

BIC ..... Bayes Information Criterion *page 58*

CHMM..... Coupled Hidden Markov Models *page 34*

CPT ..... Conditional Probability Table *page 18*

DML-HMM . . . Dynamically-multi-linked Hidden Markov Models *page 34*

EM . . . . . Expectation-Maximization *page 19*

ESS . . . . . Expected Sufficient Statistics *page 19*

GHMM . . . . . Growing Hidden Markov Models *page 90*

GNG . . . . . Growing Neural Gas *page 70*

GWR . . . . . Grow When Required *page 70*

HMM . . . . . Hidden Markov Models *page 31*

ITM . . . . . Instantaneous Topological Map *page 70*

JPD . . . . . Joint Probability Distribution *page 14*

JPDA . . . . . Joint Probabilistic Data Association *page 111*

MAP . . . . . Maximum *a Posteriori* *page 58*

MDM . . . . . Markov Dynamic Models *page 33*

ML . . . . . Maximum Likelihood *page 58*

PDF . . . . . Probability Density Function *page 14*

SOM . . . . . Self-organizing map *page 68*

TRN . . . . . Topology-representing networks *page 67*

VQ . . . . . Vector Quantization *page 34*

Bibliography

S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2): 174–188, feb 2002.

F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

F. Aurenhammer and R. Klein. *Voronoi diagrams*, chapter V, pages 201–290. Elsevier Science Publishing, 2000.

F. Bacao, V. Lobo, and M. Painho. Self-organizing maps as substitutes for k-means clustering. In *In Proc. of the Int. Conf. on Computer Science*, pages 476–483, Atlanta (US), May 2005.

Y. Bar-Shalom and T. E. Fortmann. *Tracking and data association*. Number 179 in Mathematics in science and engineering. Academic Press, Boston;London, 1988.

L. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of data*, pages 322–331, Atlantic City (US), 1990. ACM Press.

R. Bellman. *Adaptive control processes: a guided tour*. Princeton University Press, 1961.

M. Bennewitz, W. Burgard, and S. Thrun. Learning motion patterns of persons for mobile service robots. In *Proc. of the IEEE Int. Conf. On Robotics and Automation*, pages 3601–3606, Washington, USA, 2002.

M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant robot motion. *Internationl Journal of Robotics Research*, 24(1):31–48, January 2005.

H. Binsztok and T. Artières. Learning model structure from data: an application to on-line handwriting. *Electronic Letter on Computer Vision and Image Analyisis*, 5(2), 2005.

Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004. ISSN 0162-8828.

M. Brand. Structure learning in conditional probability models via an entropic prior and parameter extinction. Technical report, MERL a Mitsubishi Electric Research Laboratory, 1998.

M. Brand and V. Kettnaker. Discovery and segmentation of activities in video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):844–851, 2000.

M. Brand, N. Oliver, and A. Pentland. Coupled hidden markov models for complex action recognition. In *In Proc. of the 1997 Conf. on Computer Vision and Pattern Recognition*, pages 994–999, San Juan (PR), 1997.

A. Bruce and G. Gordon. Better motion prediction for people-tracking. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, New Orleans, US, April 2004.

H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov models. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

D. Buzan, S. Sclaroff, and G. Kollios. Extraction and clustering of motion trajectories on video. In *Proc. of the Int. Conf. on Pattern Recognition*, Cambridge, UK, 2004.

A. Caporossi, D. Hall, P. Reignier, and J. Crowley. Robust visual tracking from dynamic control of processing. In *International Workshop on Performance Evaluation of Tracking and Surveillance*, pages 23–31, Prague (CZ), May 2004.

A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: discrete bayesian models for mobile-robotnavigation. pages 963–972, Osaka (JP), 1996.

C. C. Chang and K.-T. Song. Environment prediction for a mobile robot in a dynamic environment. *IEEE Transactions on Robotics and Automation*, 13(6):862–872, 1997.

G. F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

R. T. Cox. Probability, frequency and reasonable expectation. *American Journal of Physics*, (14):1–13, 1946.

H. Dee and D. Hogg. Detecting inexplicable behaviour. In *In Proceedings of the British Machine Vision Conference*, pages 49–55, Kingston (UK), 2004.

H.-M. Dee. *Explaining Visible Behaviour*. PhD thesis, University of Leeds, 2005.

N. Dempster, A.and Laird, , and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 9(1):1–38, 1977. Series B.

D. C. Dennett. *The Intentional Stance*. The MIT Press/Bradford Books, Cambridge (US), 1987.

A. K. Dewdney and J. K. Vranch. A convex partition of r 3 with applications to crum's problem and knuth's post-office problem. *Utilitas Math*, 12:193–199, 1977.

R. A. Dwyer. *Higher-dimensional Voronoi diagrams in linear expected time*. ACM Press New York, NY, USA, 1989.

A. Elnagar and K. K. Gupta. Motion prediction of moving objects based on autoregressive model. *IEEE Transaction on Systems, Man and Cybernetics, Part A*, 28(6):803–810, 1998.

D. Filliat. *Cartographie et estimation globale de la position pour un robot mobile autonome*. PhD thesis, Université Paris 6, December 2001.

A. F. Foka and P. E. Trahanias. Predictive autonomous robot navigation. In *In Proc of the IEEE/RSJ Int. Conf. on Intelligent Robots and System*, volume 1, pages 490–495, Lausanne (CH), October 2002.

G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

D. Freitag and A. McCallum. Information extraction with hmm structures learned by stochastic optimization. In *Proc. of the Seventeenth Nat. Conf. on Artificial Intelligence and Twelfth Conf. on Innovative Applications of Artificial Intelligence*, pages 584–589, Austin, Texas, USA, July 2000. AAAI Press / The MIT Press.

N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proc. of the Fourteenth International Conference on Machine Learning*, pages 125–133, 1997.

B. Fritzke. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 1995.

S. Gong and T. Xiang. Recognition of group activities using dynamic probabilistic networks. In *In Proceedings of the Ninth IEEE International Conference on Computer Vision*, volume 2, pages 742–749, Washington (US), 2003.

A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. of the 1984 ACM SIGMOD Int. Conf. on Management of data*, pages 47–57, Boston (US), 1984. ACM Press.

R. K. Guy. Unsolved problems come of age. *The American Mathematical Monthly*, 96(10): 903–909, 1989.

K. Han and M. Veloso. Physical model based multi-objects tracking and prediction in robosoccer. In *Working notes of the AAAI 1997 Fall Symposium on Model-directed Autonomous Systems*, Boston (US), November 1997.

D. Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond (US), 1995.

R. A. Howard. *Dynamic Programming and Markov Process*. MIT Press, Cambridge (US), 1960.

W. Hu, D. Xie, and T. Tan. A hierarchical self-organizing approach for learning the patterns of motion trajectories. *IEEE Transactions on Neural Networks*, 15(1):135–144, 2004a.

W. Hu, D. Xie, T. Tieniu, and S. Maybank. Learning activity patterns using fuzzy self-organizing neural network. *IEEE Trans. on Systems, Man and Cybernetics*, 34(3):1618–1626, June 2004b.

W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank. A system for learning statistical motion patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1450–1464, September 2006.

C. Igel, T. Suttorp, and N. Hansen. A computational efficient covariance matrix update and a (1+1)-cma for evolution strategies. In *Proc. of the 8th annual conf. on Genetic and Evolutionary Computation*, pages 453–460. ACM Press New York, NY, USA, 2006.

T. Irony and N. Singpurwalla. Noninformative priors do not exist: A discussion with jose m. bernardo. *Journal of Statistical Infererence and Planning*, 65:159–189, 1997.

A. H. J. Rittscher, A. Blake and G. Stein. Mathematical modelling of animate and intentional motion. *Philosophical transactions-Royal Society of London. Biological sciences*, 358(1431): 475–490, 2003.

A. Jain, M. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31: 265–322, September 1999.

E. T. Jaynes. Probability theory- the logic of science. Unpublished manuscript, available at http: bayes.wustl.edu, 1995.

J. Jockusch and H. Ritter. An instantaneous topological map for correlated stimuli. In *In Proc. of the International Joint Conference on Neural Networks*, volume 1, pages 529–534, Washington (US), July 1999.

N. Johnson and D. Hogg. Learning the distribution of object trajectories for event recognition. In *Proc. of the British Machine Vision Conference*, volume 2, pages 583–592, September 1995.

B.-H. Juang, S. E. Levinson, and M. M. Sondhi. Maximum likelihood estimation for multivariate mixture observations of markov chains. *IEEE Transactions on Information Theory*, 32(2): 307–309, March 1986.

S. Julier and J. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *In Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, Orlando (US), 1997.

I. Junejo, O. Javed, and M. Shah. Multi feature path modeling for video surveillance. In *Proc. of the 17th conference of the International Conference on Pattern Recognition (ICPR)*, pages 716–719, 2004.

R. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:35–45, 1960.

L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series In Probability And Mathematical Statistics. John Wiley and Sons, Inc., 1989.

B. King. Step-wise clustering procedures. *Journal of the American Statistical Association*, 69: 86–101, 1967.

T. Kohonen. The 'neural' phonetic typewriter. *Computer*, 21(3):11–22, 1988. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/2.28.

T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1995. (Second Extended Edition 1997).

E. B. Koller-Meier and L. Van Gool. Modeling and recognition of human actions using a stochastic approach. In *2nd European Workshop on Advanced Video-Surveillance Systems*, Kingston, UK, September 2001.

B. J. A. Krose and M. Eecen. A self-organizing representation of sensor space for mobile robot navigation. In *In Proc. of the IEEE/RSJ/GI Int. Conf. on Intelligent Robots and Systems*, volume 1, pages 9–14, Munich (GR), 1994.

E. Kruse and F. Wahl. Camera-based observation of obstacle motions to derive statistical data for mobile robot motion planning. In *In Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 662–667, Leuven (BE), May 1998.

E. Kruse, R. Gutsche, and F. Wahl. Estimation of collision probabilities in dynamic environments for path planning with minimum collision probability. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1288–1295, 1996.

E. Kruse, R. Gusche, and F. M. Wahl. Acquisition of statistical motion patterns in dynamic environments and their application to mobile robot motion planning. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 713–717, Grenoble, France, 1997.

B. Kuipers. A hierarchy of qualitative representations for space. *Lecture Notes in Computer Science*, 1404:337–350, 1998.

P. Langley. *Learning in Humans and Machines: Towards an Interdisciplinary Learning Science*, chapter Order effects in incremental learning. Pergamon, 1995.

O. Lebeltel. *Programmation Bayéssiene des Robots*. PhD thesis, Institut National Polytechnique de Grenoble, 1999.

S. E. Levinson, L. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *Bell System Technical Journal*, 62(4):1035–1074, 1983.

Y. Li, L.-Q. Xu, J. Morphett, and R. Jacobs. An integrated algorithm of incremental and robust pca. In *Proc. of the Int. Conf. on Image Processing*, volume 1, pages 245–248, 2003.

L. Liao, D. Fox, J. Hightower, H. Kautz, and D. Schulz. Voronoi tracking: Location estimation using sparse and noisy sensor data. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, US, 2003.

Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, COM-28:84–95, 1980.

P. X. Liu, M. Meng, and C. Hu. On-line data-driven fuzzy clustering with applications to real-time robotic tracking. In *2004 IEEE Int. Conf. on Robotics and Automation*, pages 5039–5044, New Orleans, USA, April 2004.

S. P. Lloyd. Least squares quantization in pcm's. Bell Telephone Laboratories Paper, 1957.

P. Lockwood and M. Blanchet. An algorithm for the dynamic inference of hidden markov models (dihmm). In *In Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, volume 2, Minneapolis, US, April 1993.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. L. Cam and J. Neyman, editors, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

D. Magee. Tracking multiple vehicles using foreground, background and shape models. *Image and Vision Computing*, 22:143–155, 2004.

D. Makris and T. Ellis. Finding paths in video sequences. In *Proc. of the British Machine Vision Conference*, pages 263–272, 2001.

D. Makris and T. Ellis. Spatial and probabilistic modelling of pedestrian behavior. In *Proc. of the British Machine Vision Conference*, pages 557–566, Cardiff, UK, 2002.

T. Mann. Numerically stable hidden markov model implementation. An HMM scaling tutorial, 2006.

C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge (US), 1999.

S. Marsland, J. Shapiro, and U. Nehmzow. A self-organizing network that grows when required. *Neural Networks*, 2002.

T. Martinetz and K. Schulten. A "neural-gas" network learns topologies. *Artificial Neural Networks*, I:397–402, 1991.

M. Meila, M. I. Jordan, and L. P. Kaelbling. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1(1):1–48, 2001.

T. P. Minka. From hidden markov models to linear dynamical systems. Technical report, MIT, 1999.

D. C. Minnen and C. R. Wren. Finding temporal patterns by data decomposition. In *In Proc. of the Sixth IEEE Int. Conf. on Automatic Face and Gesture Recognition, 2004.*, pages 608–613, 2004.

K. P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley (USA), 2002.

R. M. Neal and G. E. Hinton. A new view of the em algorithm that justifies incremental, sparse and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998.

N. M. Oliver, B. Rosario, and A. P. Pentland. A bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8): 831–843, August 2000.

S. Osentoski, V. Manfredi, and S. Mahadevan. Learning hierarchical models of activity. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Sendai, Japan, 2004.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kauffman, 1988.

A. Pentland and A. Liu. Modeling and prediction of human behavior. *Neural Computation*, 11 (1):229–242, 1999.

L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.

R. P. N. Rao. Robust kalman filters for prediction, recognition, and learning. Technical Report TR645, The University of Rochester, 1996. URL citeseer.ist.psu.edu/rao96robust.html.

J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Symp. on the Foundations of Computer Science*, pages 144–154, Portland, US, October 1985.

R.Madhavan and C. Schlenoff. Moving object prediction for off-road autonomous navigation. In *Proceedings of the SPIE Aerosense Conference*, Orlando, FL (USA), 2003.

R. Rosales and S. Sclaroff. Improved tracking of multiple humans with trajectory prediction and occlusion modeling. In *IEEE CVPR Workshop on the Interpretation of Visual Motion*, 1998.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden markov model structure for information extraction. In *AAAI 99 Workshop on Machine Learning for Information Extraction*, Orlando, US, 1999.

Y. Singer and M. K. Warmuth. Training algorithms for hidden markov models using entropy based distance functions. In *Advances in Neural Information Processing Systems 9, NIPS*, pages 641–647, Denver, CO (USA) December 2-5, 1996, 1996. MIT Press.

C. Stauffer and E. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):747–757, August 2000.

A. Stolcke and S. Omohundro. Hidden markov model induction by bayesian model merging. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 11–18, Denver, USA, 1993. Morgan Kaufmann, San Mateo, CA.

A. Stolcke and S. M. Omohundro. Best-first model merging fr hidden markov model induction. Technical Report TR-94-003, International Computer Science Institute, Berkeley, US, 1994 1994.

L. Stone, C. Barlow, and T. Corwin. *Bayesian Multiple Target Tracking*. Artech House, 1999.

N. Sumpter and A. Bulpitt. Learning spatio-temporal patterns for predicting object behaviour. *Image and Vision Computing*, 18(9):697–704, 2000.

S. Tadokoro, M. Hayashi, Y. Manabe, Y. Nakami, and T. Takamori. Motion planner of mobile robots which avoid moving human obstacles on the basis of stochastic prediction. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 3286–3291, 1995.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

R. C. Vasko, A. El-Jaroudi, J. Boston, and T. E. Rudy. Hidden markov model topology estimation to characterize the dynamic structure of repetitive lifting data. In *In Proc. of the 19th Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Society*, pages 1725–1728, Chicago, US, October 1997.

D. Vasquez and T. Fraichard. Motion prediction for moving objects: a statistical approach. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 3931–3936, New Orleans, LA (US), April 2004. URL http://emotion.inrialpes.fr/bibemotion/2004/VF04.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, April 1967.

M. Walter, A. Psarrow, and S. Gong. Learning prior and observation augmented density models for behaviour recognition. In *Proc. of the British Machine Vision Conference*, pages 23–32, 1999.

X. Wang, K. Tieu, and E. Grimson. Learning semantic models by trajectory analysis. Technical report, Massachussets Institute of Technology, 2006.

T. Xiang and S. Gong. Beyond tracking: Modelling activity and understanding behaviour. *International Journal of Computer Vision*, 2006.

H. Yu and T. Su. Destination driven motion planning via obstacle motion prediction and multistate path repair. *Journal of Intelligent and Robotic Systems*, 36(2):149–173, February 2003.

Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

Q. Zhu. A stochastic algorithm for obstacle motion prediction in visual guidance of robot motion. *The IEEE International Conference on Systems Engineering*, pages 216–219, 1990.