KTH Computer Science
and Communication

Intention Recognition in Human Machine Collaborative Systems

DANIEL K. E. AARNO

Licentiate Thesis
Stockholm, Sweden 2007

TRITA CSC-A 2007:2
ISSN 1653-5723
ISRN KTH/CSC/A--07/02--SE
ISBN 978-91-7178-581-7

KTH School of Computer Science and Communication
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie licentiatexamen i datalogi den 23 mars, 2007 i sal D31, Kungl Tekniska högskolan, Lindstedtsv 17, Stockholm.

Tryck: Universitetsservice US AB

Abstract

Robot systems have been used extensively during the last decades to provide automation solutions in a number of areas. The majority of the currently deployed automation systems are limited in that the tasks they can solve are required to be repetitive and predicable. One reason for this is the inability of today's robot systems to understand and reason about the world. Therefore the robotics and artificial intelligence research communities have made significant research efforts to produce more intelligent machines. Although significant progress has been made towards achieving robots that can interact in a human environment there is currently no system that comes close to achieving the reasoning capabilities of humans.

In order to reduce the complexity of the problem some researchers have proposed an alternative to creating fully autonomous robots capable of operating in human environments. The proposed alternative is to allow *fusion* of human and machine capabilities. For example, using teleoperation a human can operate at a remote site, which may not be accessible for the operator for a number of reasons, by issuing commands to a remote agent that will act as an extension of the operator's body.

Segmentation and recognition of operator generated motions can be used to provide appropriate assistance during task execution in teleoperative and human-machine collaborative settings. The assistance is usually provided in a virtual fixture framework where the level of compliance can be altered online in order to improve the performance in terms of execution time and overall precision.

Acquiring, representing and modeling human skills are key research areas in teleoperation, programming-by-demonstration and human-machine collaborative settings. One of the common approaches is to divide the task that the operator is executing into several sub-tasks in order to provide manageable modeling.

This thesis is focused on two aspects of human-machine collaborative systems. *Classification* of an operator's motion into a predefined state of a manipulation task and assistance during a manipulation task based on *virtual fixtures*. The particular applications considered consists of manipulation tasks where a human operator controls a robotic manipulator in a cooperative or teleoperative mode.

A method for online task tracking using *adaptive virtual fixtures* is presented. Rather than executing a predefined plan, the operator has the ability to avoid unforeseen obstacles and deviate from the model. To allow this, the probability of following a certain trajectory (sub-task) is estimated and used to automatically adjusts the compliance of a virtual fixture, thus providing an online decision of how to fixture the movement.

A layered hidden Markov model is used to model human skills. A gestem classifier that classifies the operator's motions into basic action-primitives, or gestemes, is evaluated. The gestem classifiers are then used in a layered hidden Markov model to model a simulated teleoperated task. The classification performance is evaluated with respect to noise, number of gestemes, type of the hidden Markov model and the available number of training sequences. The layered hidden Markov model is applied to data recorded during the execution of a trajectory-tracking task in 2D and 3D with a robotic manipulator in order to give qualitative as well as quantitative results for the proposed approach. The results indicate that the layered hidden Markov model is suitable for modeling teleoperative trajectory-tracking tasks and that the layered hidden Markov model is robust with respect to misclassifications in the underlying gestem classifiers.

Sammanfattning

Robotsystem har använts flitigt under de senaste årtiondena för att skapa automationslösningar i ett flertal områden. De flesta nuvarande automationslösningarna är begränsade av att uppgifterna de kan lösa måste vara repetitiva och förutsägbara. En av anledningarna till detta är att dagens robotsystem saknar förmåga att förstå och resonera om omvärlden. På grund av detta har forskare inom robotik och artificiell intelligens försökt att skapa intelligentare maskiner. Trots att stora framsteg har gjorts då det gäller att skapa robotar som kan fungera och interagera i en mänsklig miljö så finns det för nuvarande inget system som kommer i närheten av den mänskliga förmågan att resonera om omvärlden.

För att förenkla problemet har vissa forskare föreslagit en alternativ lösning till helt självständiga robotar som verkar i mänskliga miljöer. Alternativet är att *kombinera* människors och maskiners förmågor. Exempelvis så kan en person verka på en avlägsen plats, som kanske inte är tillgänglig för personen i fråga på grund av olika orsaker, genom att använda fjärrstyrning. Vid fjärrstyrning skickar operatören kommandon till en robot som verkar som en förlängning av operatörens egen kropp.

Segmentering och identifiering av rörelser skapade av en operatör kan användas för att tillhandahålla korrekt assistans vid fjärrstyrning eller samarbete mellan människa och maskin. Assistansen sker ofta inom ramen för virtuella fixturer där eftergivenheten hos fixturen kan justeras under exekveringen för att tillhandahålla ökad prestanda i form av ökad precision och minskad tid för att utföra uppgiften.

Den här avhandlingen fokuserar på två aspekter av samarbete mellan människa och maskin. Klassificering av en operatörs rörelser till ett på förhand specificerat tillstånd under en manipuleringsuppgift och assistans under manipuleringsuppgiften baserat på virtuella fixturer. Den specifika tillämpningen som behandlas är manipuleringsuppgifter där en mänsklig operatör styr en robotmanipulator i ett fjärrstyrt eller samarbetande system.

En metod för att följa förloppet av en uppgift medan den utförs genom att använda virtuella fixturer presenteras. Istället för att följa en på förhand specificerad plan så har operatören möjlighet att undvika oväntade hinder och avvika från modellen. För att möjliggöra detta estimeras kontinuerligt sannolikheten att operatören följer en viss trajektorie (deluppgift). Estimatet används sedan för att justera eftergivenheten hos den virtuella fixturen så att ett beslut om hur rörelsen ska fixeras kan tas medan uppgiften utförs.

En flerlagers dold Markovmodell (eng. layered hidden Markov model) används för att modellera mänskliga färdigheter. En gestemklassificerare som klassificerar en operatörs rörelser till olika grundläggande handlingsprimitiver, eller gestemer, evalueras. Gestemklassificerarna används sedan i en flerlagers dold Markovmodell för att modellera en simulerad fjärrstyrd manipuleringsuppgift. Klassificeringsprestandan utvärderas med avseende på brus, antalet gestemer, typen på den dolda Markovmodellen och antalet tillgängliga träningssekvenser. Den flerlagers dolda Markovmodellen tillämpas sedan på data från en trajektorieföljningsuppgift i 2D och 3D med en robotmanipulator för att ge både kvalitativa och kvantitativa resultat. Resultaten tyder på att den flerlagers dolda Markovmodellen är väl lämpad för att modellera trajektorieföljningsuppgifter och att den flerlagers dolda Markovmodellen är robust med avseende på felklassificeringar i de underliggande gestemklassificerarna.

Acknowledgments

First of all I would like to thank my supervisor **Danica Kragić** for providing me with the opportunity to pursue this work. Thank you for all the proof reading, editing, general support and keeping my spirit up by pushing, pulling or prodding me when required.

A big thank you goes to all my friends at CAS and CVAP for providing a pleasant working atmosphere and stimulating "coffee-break" discussions. I would especially like to thank:
Staffan Ekvall for all the collaboration, office discussions and for providing a pleasant working atmosphere by arranging soccer matches, video-game evenings, and letting know about all your "get-rich-quick-schemes".
Frank Lingelbach for all the discussions and MATLAB help. Thank you for all the barbecues, parties and for helping me to maintain the Thursday-pizza tradition.
Andreas Hedström for never getting bored with all the pointless Linux and programming discussions.
Patric Jensfelt also deserves a special thank's for helping with all the hardware and software and generally keeping the lab up and running.

I would also like to thank all my undergraduate study-buddies, and especially the "magnificent seven" **Gohde**, **Stefan**, **Styrsel**, **Tower**, **Wallin** and **Wincent** for making my years at MDH and KTH so much more fun and interesting.

My **family** also deserves acknowledgment for supporting me and providing me with a stable base which I know I can always rely on if I need it.

Finally I would like to thank the Swedish tax payers for supporting this work through **Vetenskapsrådet**.

Contents

List of Figures

Chapter 1

Introduction

During the last decades robots have become a common commodity in the industry. Robots have been extensively used in the areas of automotive production, for foundry and forging, for packaging and palletizing as well as in metal fabrication and production of plastics. Traditionally the requirements for successful deployment of robotic systems have been that they should deal with mass production where the robot's task is well defined and should be performed repeatedly without the possibility of adapting to changes in the process. Robotics are continuously make ways into new areas and are now being used for, among other things, domestic tasks, surgery, surveillance and for disposal of bombs and other hazardous materials.

One endeavor in the robotics research community is to try to equip robots with some level of artificial intelligence in order to allow robots to be deployed in a variety of settings where the current robots fails because of their inability to understand the world and adapt to changes in their surrounding. This is a daunting undertaking and even though progress is constantly made towards robots capable of interacting in human environments there are presently no robotic systems that come close to displaying the reasoning capabilities or ingenuity of humans when it comes to detecting and handling unforeseen events. This has lead researchers in a new direction seeking a collaboration between humans and robots in order to solve tasks that are beyond the human or robot alone.

One area where collaboration between humans and robots has already begun to bear fruit is in the manufacturing industry where large portion of the procedures have been automated. However, many processes are too difficult to automate and must rely on humans' supervisory control and decision making; in areas such as the identification of defective parts and process variations. When such skills are required, humans still have to perform straining tasks. Therefore, human-machine collaborative systems (HMCS) has been used to prevent ergonomic injuries and operator wear, by allowing cooperation between a human and a robotic system in a flexible way (Peshkin *et al.*, 2001; Moore Jr. *et al.*, 2003; Schraft *et al.*, 2005).

An other area where robotics is currently emerging is for surgical procedures (Taylor and Stoianovici, 2003). The application of robots to surgical procedures range from "smart" tools to fully autonomous procedures carried out by a robot after pre-operative planning by a surgeon, see (Dario *et al.*, 2003) for a survey.

There are two areas where robots are expected to have the highest impact in surgery. These areas are minimal invasive surgery (MIS) and microsurgery. Microsurgical tasks are difficult for surgeons simply because of the scale at which the surgeon must operate. The accuracy required to perform some of these procedures require accurate positioning of a tool tip within 10 μm of a target. Achieving such high accuracy is very difficult for a surgeon. A common example of a microsurgical task that is beyond the existing manual techniques is the injection of anticoagulants in a retinal vein (Riviere *et al.*, 2003). Using active devices such as robots or active tools it should be possible to perform surgery on these scales.

In MIS the challenge is to provide a natural operating environment for the surgeon. With today's systems the surgeon usually has poor haptic presences and is limited with respect to the number of DOF of the endoscopic tools. A teleoperated setting could be used here in order to provide the surgeon with motion and force scaling or even help the surgeon prevent unintentional damage to surrounding tissue by actively monitoring the positioning of the tool avoiding any forbidden regions specified during preoperative planning. Using a teleoperative setting also provides the possibility of enhanced ergonomics for the surgeon.

Learning human skills, using them in a HMCS or transferring them to robots directly has been a core objective for more than three decades in the area of artificial intelligence, robotics and intelligent control. Application areas range from teleoperation to programming-by-demonstration (PbD), human-machine collaborative settings automated visual surveillance and multi-modal human-computer interaction (Kaiser and Dillmann, 1996; Liang and Ouhyoung, 1998; Hundtofte *et al.*, 2002; Zöllner *et al.*, 2002; Li and Okamura, 2003; Elgammal *et al.*, 2003; Castellani *et al.*, 2004; Oliver *et al.*, 2004; Kragić *et al.*, 2005; Yu *et al.*, 2005).

It has been widely recognized that the underlying system used for learning, representing, modeling and transferring of skills has to deal with highly nonlinear relationships between the stimuli and response. In addition such a system is strongly dependent on the varying state of the environment and the user that performs them.

One idea that has received much attention lately is that if the intention of an operator of a teleoperated system can be recognized online in real-time, it is possible to improve the task execution by allowing the system to adapt to the operator's need by applying the correct control mode in the transfer step. To be able to give the correct aid to the operator it is necessary for the HMCS to be able to successfully interpret the operator's intent, online and in real-time. For example, medical robots increase the performance with their superior precision but are still not capable of safe decision-making.

This chapter briefly describes the general concept of a HMCS and gives examples of applications. It then describes the outline and contributions of this thesis.

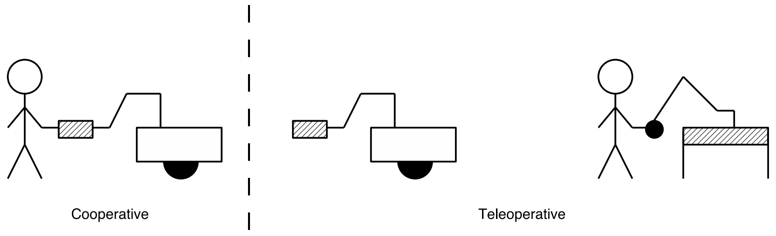


Figure 1.1. Cooperative (left) and teleoperative (right) systems.

1.1 Human Machine Collaborative Systems

In a human-machine collaborative system (HMCS) the machine is supposed to increase the performance of the human operator by providing assistance. At the same time the operator can increase the performance of the machine by allowing a much wider range of tasks to be solved than in an autonomous system. Thus assistance in a HMCS works both ways, the machine augments the operator's capabilities with its superior precession, repeatability and endurance. On the other hand the operator helps the machine with difficult high-level decision making such as error recovery and handling of task deviations.

In this thesis we focus on two specific aspects of HMCS. Recognition of the operator's intent and assistance based on *virtual fixtures*. Although our methods are not limited to such applications we concentrate on applications consisting of manipulation tasks where a human operator controls a robotic manipulator in a cooperative or teleoperated mode. The difference between cooperative and teleoperative mode should be addressed at this point. In *cooperative* mode the robot and human are *physically linked*. That is, both the human and robot are in direct contact to the end-effector by, for example, holding the same workpiece or directly applying forces to the end-effector by some other mechanism. In a *teleoperated* system there is no such physical connection and the human can only control the master robot through a *slave device* which may or may not have the same kinematics as the master, be able to provide force feedback etc, as illustrated in figure 1.1.

1.1.1 Humans Assisting Machines

Humans' ability to perceive and reason about the world is far superior to what the robotics community has achieved so far. Especially when it comes to dealing with error recovery, unexpected situations and decision making under uncertainty, human beings achieve much better results than any intelligent robot system presented to date. Thus by allowing human interference during the execution of autonomous tasks it is possible to solve tasks that autonomous robots cannot deal with on their own. The simplest form of assistance can be to switch the autonomous system to manual control when an error is detected that the autonomous system does not know how to handle. Thus during normal operation the system works in autonomous mode, but once an error the system is unable to handle is detected, the execution of the autonomous plan is suspended and human assistance is requested. This can also work the other way around, that is, a human operator monitors the autonomous system and intervenes by assuming manual control if the autonomous system is about to perform an undesired, possibly dangerous, action.

More intelligent cooperation can be achieved by having the human operator and the robot *share control*. This means that some degrees of freedom (DOF) are controlled by the robot and some by a human operator. A typical example is shared position/force control where the robot's control system controls contact forces while the human operator controls the position and orientation of the end effector (Bruyninckx and Schutter, 1997).

Another way humans can assist machines is by providing high-level commands, i.e. specifying an operational plan, that the robot carries out autonomously. An example would be that the operator specifies the action sequence, `goto table; pick up cup; goto kitchen; put cup in dish washer` Which is then carried out autonomously by the agent. Thus the high level plan is performed by the human that has the capabilities to reason about the world and realize that there is a dirty cup on the table that need to be brought to the dish washer. The robot can then carry out this plan autonomously, given that it knows how to perform the required actions.

1.1.2 Machines Assisting the Human

Machines can be used to assist humans with their superior precession and endurance. For example force scaling can be used to amplify contact forces at the tool-point of a tool simultaneously operated by the human and the robot. Using force scaling it is possible for the human operator to produce less contact forces and improve the execution of the task by receiving better feedback. The forces can be scaled differently along different dimensions depending on the task so that for example forces perpendicular to a surface are amplified less then forces in the surface plane. Motion scaling is also possible in the event of a teleoperated setting. That is, the motion induced by the human at the slave device is scaled at the tool-point. The motion can be scaled up or down depending on the task. Motion scaling allows humans to perform tasks at scales where normal human capabilities are insufficient, such as micro-surgical tasks. Motion and force scaling can be combined to improve the human's performance by providing more suitable feedback than during direct execution of the task.

To provide smooth and safe control in medical collaborative settings, a careful teleoperative design has to be provided. If the intent of the operator can be understood the performance can be increased by applying the correct type of force scaling and control modes (Li and Okamura, 2003).

Another way in which machines can assist humans is by tremor reduction. The operator's input can be filtered to remove small random motions that occur due to tremor, before the motion commands are sent to the tool. Robots can also be used to effectively remove the weight and inertia of heavy objects being manipulated. By examining the forces produced by a human operator, on an object simultaneously held by the robot and human, the robot can act in such way that the inertia and weight of the manipulated object is effectively "removed" from the viewpoint of the human operator.

Robots can also be used for flexible fixturing (holding) of workpieces during assembly like tasks, effectively providing "extra hands" for the operator, with the additional advantage of being free from tremor and not suffering from fatigue.

1.2 Outline

In this thesis we present work on two aspects of HMCS, i) classification of an operator's motion into a predefined state of a manipulation task and ii) assistance during a manipulation task based on virtual fixtures. The particular applications considered consists of manipulation tasks where a human operator controls a robotic manipulator in a cooperative or teleoperative mode.

The contributions of this work is the proposed LHMM structure for motion intention recognition and the associated evaluation (chapter 4) and the proposed method of using the probability that the operator is executing a certain state to adjust the compliance of a virtual fixture (chapter 3). This thesis is organized in the following way.

Chapter 2

Chapter 2 briefly introduces the two problems of *intention recognition* and *assistance*. It goes on to provide a theoretical foundation for the various methods applied used in this thesis. This chapter is concluded by providing examples of previous work in the area of HMCS that is directly related to to the work presented in this thesis.

Chapter 3

It has been demonstrated in a number of robotic areas how the use of *virtual fixtures* improves task performance both in terms of execution time and overall precision. However, the fixtures are typically inflexible, resulting in a degraded performance in cases of unexpected obstacles or incorrect fixture models. In chapter 3, we propose the use of *adaptive virtual fixtures* that can cope with the above problems.

A teleoperative or human-machine collaborative setting is assumed, with the core idea of dividing the task that the operator is executing into several sub-tasks. The operator may remain in each of these sub-tasks as long as necessary and switch freely between them. Hence, rather than executing a predefined plan, the operator has the ability to avoid unforeseen obstacles and deviate from the model. In our system, the probability that the user is following a certain trajectory (sub-task) is estimated and used to automatically adjusts the compliance. Thus, an online decision of how to fixture the movement is provided.

Chapter 4

In chapter 4 we consider the use of a Layered Hidden Markov Model (LHMM) to model human skills. We evaluate a gestem classifier that classifies motions into basic action-primitives, or *gestemes*. The gestem classifiers are then used in a LHMM to model a simulated teleoperated task. We investigate the classification performance with respect to noise, number of gestemes, type of HMM and the available number of training sequences. We also apply the LHMM to data recorded during the execution of a trajectory-tracking task in 2D and 3D with a robotic manipulator in order to give qualitative as well as quantitative results for the proposed approach.

Chapter 5

This chapter summarizes the thesis and provides a discussion on possible improvements and future work.

The work presented in this thesis has been presented at international conferences and has been published in the following articles.

D. Aarno and D. Kragić *Layered HMM for Motion Intention Recognition* In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp 5130 - 5135, 2006.

S. Ekvall, D. Aarno and D. Kragić *Online Task Recognition and Real-Time Adaptive Assistance for Computer-Aided Machine Control* In *IEEE Transactions on Robotics*, 22:pp 1029 - 1033, 2006.

D. Aarno, S. Ekvall and D. Kragić *Adaptive Virtual Fixtures for Machine Assisted Teleoperation Tasks* In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp 897 - 903, 2005.

In addition a number of publications not covered in this thesis have been produced during the scope of these studies.

D. Aarno, J. Sommerfeld, D. Kragić, N. Pugeault, S. Kalkan, F. Wörgötter, D. Kraft and N. Krüger *Model-independent grasping initializing object-model learning in a cognitive architecture* In *IEEE International Conference on Robotics and Automation, Workshop: "From features to actions - Unifying perspectives in computational and robot vision"*, 2007 [to appear].

P. Jensfelt, S. Ekvall, D. Kragić and D. Aarno *Augmenting SLAM with Object Detection in a Service Robot Framework* In *Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication*, pp 741 - 746, 2006.

S. Ekvall, D. Aarno and D. Kragić *Task Learning Using Graphical Programming and Human Demonstrations* In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication* pp 398 - 403, 2006.

P. Jensfelt, S. Ekvall, D. Kragić and D. Aarno *Integrating SLAM and Object Detection for Service Robot Tasks* In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Workshop: "Mobile Manipulators: Basic Techniques, New Trends and Applications"*, 2005.

D. Aarno, F. Lingelbach and D. Kragić, *Constrained Path Planning and Task-Consistent Path Adaptation for Mobile Manipulators* In *Proceedings of the International Conference on Advanced Robotics* pp 268 - 273, 2005.

D. Kragić, S. Ekvall, P. Jensfelt and D. Aarno *Sensor Integration and Task Planning for Mobile Manipulation* In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Workshop: "Issues and Approaches to Task Level Control"*, 2004.

Chapter 2

Background and Related Work

The scenario considered in this thesis is that of a HMCS where a human and a robot interacts to solve a task with a common goal. Furthermore the robot should try to estimate the intention of the human in order to provide better assistance. We specially focus on manipulation tasks that can be subdivided into smaller tasks consisting of specific motions.

To accomplish this there are two key problems that must be solved.

- Recognition of the human's intent in order to identify the current state of the task.
- Provide useful assistance depending on the state of the task.

This chapter first briefly introduces the two problems of *intention recognition* and *assistance*. It goes on to provide a theoretical foundation for the various methods applied in chapter 3 and 4. Once the required theoretical background is covered this chapter is concluded by providing examples of previous work in the area of HMCS that is directly related to to the work presented in this thesis.

Intention Recognition

In order for robots to cooperate nicely with humans or other robots, hereafter agents, in a shared environment it is important that they are able to interpret the actions and estimate the intentions of these other agents. If a robot, or any other system, is able to recognize the intention of other agents it will have a better possibility to plan ahead and adapt its behavior to better suit other agents.

For a system to be able to recognize the intention of an agent it is necessary to be able to *classify* the agents actions and relate them to an internal model of the task. Classification of data has been extensively dealt with in the area of Machine Learning (ML). For the case of recognition of an agent's intent it will be necessary to deal with the three closely related problems of *sequential supervised learning, time-series analysis* and *sequence classification*, (Dietterich, 2002).

Intention recognition is not limited to dealing with robot motions. The following examples from the literature illustrates applications in other areas.

In the work of Dielmann and Renals (2004) the goal is to analyze meetings in order to be able to classify various meeting actions. The actions considered in (Dielmann and Renals, 2004) were monologue, dialog, note taking, presentation and presentation at the white board during meetings with four people. The sensors used where one lapel microphone per meeting participant and a central circular microphone array with eight microphones. Using the microphone array sound directions could be estimated. It was assumed that meeting participants would either be in their seat, presenting or presenting at the white board. Consequently six "speech activities" where considered. The features used for training and recognition where the estimated speaker activities during the last three time steps, forming a 216 dimensional feature vector and prosodic features extracted from the lapel microphones forming a 12 dimensional feature vector. A two level HMM approach was used to classify the meeting data using either *early integration*, where the different feature vectors are concatenated, or a multi stream approach where the different feature types are first classified independently and then integrated at in the top level HMM.

Shimosaka *et al.* (2005) used switching linear dynamics with marginalized bags of vector kernels to classify human actions into walking and non-walking. The input to the classifier consisted of human motion data. The motion data was comprised of 36 values describing the skeletal configuration of the human. The motions were classified into two categories, walking and non-walking. The walking category contained examples with varying tempo and the non-walking category contained examples from sitting still, lying still, standing still, running and translational motion from standing to sitting. The classification was performed online.

A different application of motion classification is presented by Lin *et al.* (2005). The idea is to use motion classification during MIS tasks in order to estimate the quality of the performed task, for example during training of surgeons. A model of the task as a sequence of elementary suturing motions is extracted and linear discriminant analysis is used to separate motions from different gestures. It is shown that the motions of an expert surgeon separates better than the motions of a less experienced surgeon.

Mori *et al.* (2004) applied HMMs at various level to perform recognition of daily human actions such as standing still, sitting and walking. A tree structure was used so that recognition could take place at various levels of detail. For example the action would first be classified into sitting, lying or standing and, if for example the action was classified to sitting, it would be classified into sitting on a chair or sitting on the floor. Using a tree structure has two advantages. It makes the recognition problem simpler because some irrelevant features can be excluded at the detailed levels and it is possible to give reasonable responses to novel data by only applying coarse classification.

A two-layer HMM was used by Zhang *et al.* (2004) to model individual and group actions during meetings. An I-HMM was used to model individual actions.

The recognized individual actions was then passed along to the G-HMM that was used to classify group actions.

Oliver *et al.* (2004) used a LHMM to recognize different types of activity in an office environment. Xie *et al.* (2003) used a HHMM to automatically segment a soccer game into two classes, *pause* and *play* in an unsupervised setting.

Assistance

Assistance provided by a robot to a human operator can take several forms and be interpreted in various ways. In the following assistance implies that robot and human are working together in a shared workspace and are collaboratively controlling an end-effector or a workpiece. That is, the robot is not autonomous since it requires input from the human operator, nor is it a slave device simply executing the operator's instructions. Thus for a robot to be able to provide assistance it is necessary to incorporate task knowledge in the control scheme. The following examples from the literature illustrates possible applications and methods for robots assisting humans.

Riviere *et al.* (2003) has developed a handheld surgical tool that can measure its own motion and assist the surgeon by reducing the tremor at the tool tip by actively compensating for the surgeon's tremor by deflecting its tip. The tremor was canceled using a nonlinear adaptive noise canceling algorithm based on the weighted-frequency Fourier linear combiner. The system currently only handles tremor, which is a rhythmic sinusoidal movement. The system is unable to handle non-rhythmic involuntary movements, such as jerks. There is ongoing work to extend the tool to be able to assist the surgeon by ignoring this type of involuntary motions as well.

Itoh *et al.* (2000) proposed an control algorithm for teleoperation based on *virtual tool dynamics*. Using virtual tool dynamics the motion and force of the slave manipulator is scaled in order to provide the master manipulator with virtual tool dynamics. This means that the master manipulator is controlled as if the operator was using a passive tool designed to solve the particular task at hand. The operator can select suitable virtual tools in order to have the teleoperative system provide assistance during all phases of the task.

Payandeh and Stanisic (2002) used virtual fixtures to, among other things, provide visual cues, generate and restrict motion of the robot and tune low-level control parameters. This is applied to a tele-operated acquire task where the robot must be positioned in order to approach, grasp and extract an object.

In (Moore Jr. *et al.*, 2003; Peshkin *et al.*, 2001) Cobots are presented. Cobots are used in the manifacturing industry and are collaborative devices that can be used to constrain the motion of a work piece to, for example, virtual paths or surfaces.

Bettini *et al.* (2004) used *virtual fixtures* (see section 2.2) to assist a human operator to perform path following and target approach. Virtual fixtures were used to constrain the motion to a sequence of cylindrical tunnels and cones.

Woern and Laengle (2000) presents a control scheme that allows cooperation between humans and a (semi-) autonomous robot. The robot usually operates in autonomous mode, but has the possibility to switch to a semi-autonomous mode if it detects an error or if there is missing information. In such a case a human operator is supposed to assist the robot until the problems are resolved. Once the problems are resolved autonomous execution can be resumed. The human operator also has the possibility to switch the system into the semi-autonomous mode at any time by interfering with the task.

The type of task considered in (Woern and Laengle, 2000) is a pick and place task with a mobile robot with two PUMA 560 arms. In the semi-autonomous mode the human operator is responsible for motion along some DOF while the robot remains in control of the remaining DOF. For example the human can be required to move the robot hand to the correct position using the force-torque sensor mounted on the end-effector while the robot controls the orientation of the workpiece currently being manipulated.

Guo *et al.* (1995) used event-based planning to allow fusion of human and machine intelligence. This means that if an obstacle would appear along the planned path the robot would stop and the error would remain constant. This is different from a time parametrized plan, where the error would increase. At any time during execution of the plan, human intelligence can be introduced in the system through an input device, providing fusion of the human's and robot's plans. This idea is evaluated on a system with a PUMA 560 manipulator on two tasks. The first task is avoiding an unexpected obstacle. The autonomous plan is halted because an unexpected obstacle is present along its path. A human operator introduces additional knowledge into the system by specifying that the system, in addition to following its plan, should move perpendicular to its reference direction. The perpendicular motion specification is one out of four possible actions that can be introduced into the system: stop, slow down, speed up and orthogonal motion. The other task is a hybrid position/force control where the autonomous controller maintains contact forces while the human operator controls the position and orientation of the end-effector.

Li and Taylor (2004) used virtual fixtures to improve nose surgery. A 3D model of the nose cavity was obtained from a CT-scan. The fixtures aided the surgeon in following a precomputed trajectory while assuring that boundary constraints are not violated.

The next section will explain about the various machine learning and classification algorithms used for intention recognition in this thesis. If you have a strong foundation in machine learning you may wish to skip past some parts of the next section.

2.1 Machine Learning and Classification

A *classifier* in the traditional supervised learning sense is a function $h : \mathcal{X} \rightarrow \mathcal{Y}$ that maps from a datum $x \in \mathcal{X}$ to a class $y \in \mathcal{Y}$. A training example (x, y) is a pair consisting of a datum x and its corresponding class label y taken from a set of possible class labels $\mathcal{Y}$. The training examples are usually assumed to be drawn independently and identically (iid) from a joint distribution $P(x, y)$. The training data consists of N such examples. In supervised learning the classifier h undergoes a learning process where the goal is to find an h that correctly classifies the class $y = h(x)$ of new unseen data. This is done by searching some space $\mathcal{H}$ of possible classifiers.

For motion intention recognition classical supervised learning classifiers fail because the data associated with human motions are inherently sequential in their nature. This means that the training data consists of sequences of (x, y) pairs rather than isolated pairs being drawn iid. Furthermore these sequences usually exhibit strong sequential correlation.

One way to make classification algorithms work with sequential data is to group data over time and perform classification of the grouped data. This is easily implemented as a sliding window of length L where the data in the time interval $[t-L, t]$ is passed as a datum point to the classifier. This works well in many settings and is simple to implement. However there are also problems with this approach. The window size L may affect classification performance and the optimal size may not be constant in time. In addition spectral leakage may occur as a result of windowing (Harris, 1978). Therefore it is often better to use classifiers and learning algorithms that has been especially developed to work with sequential data.

In the *sequential supervised learning problem* the training data is a set $\Gamma = \{(\mathbf{x}_i, \mathbf{y}_i)\}, \; \forall i \in [1..N]$ of N training examples. Each example is a pair $(\mathbf{x}_i, \mathbf{y}_i)$ of sequences, where $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,T}\}$ and $\mathbf{y}_i = \{y_{i,1}, y_{i,2}, \ldots, y_{i,T}\}$. The goal is to construct a classifier h that maps an unseen input sequence $\mathbf{x}$ to the correct output sequence $\mathbf{y}$. The two closely related problems mentioned earlier are time-series analysis and sequence classification.

In *sequence classification* the goal is to predict a single label to an entire sequence of input data. That is the function $y = h(\mathbf{x}_i)$ maps from a sequence of input data $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,T}\}$ to a single output class y.

For *time-series prediction* only a partial observation of $\mathbf{x}_i$ up to time t is given along with the corresponding correct class labels $\mathbf{y}_i$. The goal is then to predict the future observations of $\mathbf{x}_i$ and $\mathbf{y}_i$.

2.1.1 Markov Models

A Markov model is a model of a process that has the Markov property. The Markov property means that the probability of changing from state s at time t to state s' at time $t + 1$ depends *only on the current state* s. That is the probability $P(s'|s)$ is independent of the time t as well as any state transitions prior to time $t + 1$.

Figure 2.1 shows an example of a Markov model as a directed graph. The arrows connecting the states shows the probability of transition from one state to another. These probabilities are often stored in a *state transition probability matrix* $\mathbf{A}$, where $\mathbf{A}_{i,j}$ is the probability of transition from state i to state j. Many variants of the Markov model exists. For example there are continuous time versions where the time is updated as a continuous variable rather than in steps. There are also higher order Markov models such that for a 2nd order Markov model the state transition does not only depend on the current state but also the previous state. For a Nth order Markov model the state transition depends on the chain of state transitions over the last N time steps. Markov processes are fully observable, meaning that it is always possible to measure the current state of the process reliably. A simple example of a Markov model could be the weather. Of course the weather is not truly a Markov process since it is not possible to measure the state precisely. However, an approximation can be used. In this simple example there are only three types of weather, sunny, cloudy and raining enumerated as follows:

Weather	State
Sunny	1
Cloudy	2
Raining	3

The state transition probability matrix for the simplified weather model would look like:

$$\mathbf{A} = \begin{bmatrix} 0.8 & 0.15 & 0.05 \\ 0.2 & 0.5 & 0.3 \\ 0.05 & 0.5 & 0.45 \end{bmatrix}$$

That is, given that it is sunny today the probability that it will be sunny tomorrow is 80%. The probability that the weather will change to cloudy is 15% and that it will start raining is only 5%. The corresponding graph is shown in figure 2.1. The weather model is a *fully connected* model which means that it is possible to transition from any given state to any other state. This can be seen since there are no zero elements in the $\mathbf{A}$ matrix. The Markov model can be constrained in many ways to simplify the model. One commonly used structure is the *sequential left to right* (SLR) structure where it is only possible to transition to either the current state or the next state. That is $\mathbf{A}_{i,j} = 0, \; \forall i \neq j, \; j \neq i + 1$. Another common structure also allows stepping backwards so that $\mathbf{A}_{i,j} = 0, \; \forall i \neq j, \; j \neq i \pm 1$.

2.1.2 Hidden Markov Models

The hidden Markov model (HMM) is very similar to the Markov model described previously. However, in the HMM it is not possible to observe the current state, it is *hidden*. Compare this to the Markov model where it is always possible to observe the current state exactly. Since it is not possible to directly observe the current state in the HMM how can it be of any use? The answer is simple, it is possible to observe something *about* the current state. That is, each state is associated with

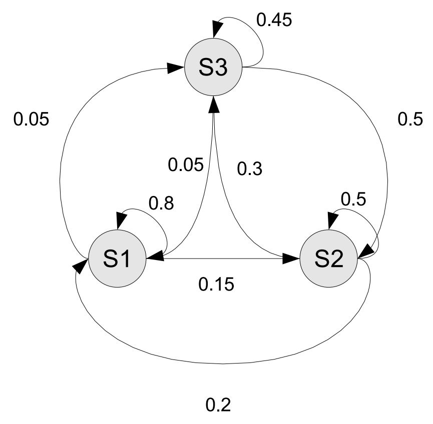


Figure 2.1. A Markov model of the weather

a set of possible *observations*. However, these observations can only be associated with a state in a statistical sense. The HMM is thus a doubly stochastic process. There is an underlying, unobservable, Markov model which can be associated with observations through observation probabilities. That means that each state in the HMM has associated with it the probability of observing a particular observation. In the simplest form of the HMM the observations are all taken from some finite enumerated set $\mathcal{O} = \{O_1, O_2, \ldots, O_M\}$. This means that it is possible to represent the observation symbol probability as a matrix $\mathbf{B}$ where $\mathbf{B}_{i,j}$ is the probability to observe the jth observation symbol in the state i. In addition, since it is generally unknown even in which state the model is at time $t = 0$ an initial state probability vector is also required. This is usually denoted by π such that π_i is the probability of starting in state i at time $t = 0$. Thus the HMM $\lambda = \{\mathbf{A}, \mathbf{B}, \pi\}$ is defined by three elements over N states and M discrete observation symbols.

- $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the state transition probability matrix. Where $\mathbf{A}_{i,j}$ is the probability of taking the transition from state i to state j.

- $\mathbf{B} \in \mathbb{R}^{N \times M}$ is the observation probability matrix, with $\mathbf{B}_{i,j}$ is the probability, $P(O_j|\text{state } i)$, of observing the jth possible observation symbol out of the total M discrete observation symbols in state i.

- $\pi \in \mathbb{R}^N$ is the initial state probability vector, where π_i is the probability of starting in state i.

The majority of applications of HMMs have been in speech recognition (Rabiner, 1989) but successful results are also reported in many other fields. When dealing with HMMs there are three problems that commonly has to be solved.

1. Given a HMM $\lambda = \{\mathbf{A}, \mathbf{B}, \pi\}$ and a sequence of observations symbols $\mathbf{o} = \{o_1, o_2, \ldots, o_T\}$ up to time T, how can the probability $\mathrm{P}(\mathbf{o} \mid \lambda)$ be computed. The probability $\mathrm{P}(\mathbf{o} \mid \lambda)$ reveals information about how likely the observations were generated from a process modeled by λ. This can be useful to determine which of several processes occurred. This is hereafter referred to as the problem of *evaluation*.

2. Given a HMM $\lambda = \{\mathbf{A}, \mathbf{B}, \pi\}$ and a sequence of observations symbols $\mathbf{o} = \{o_1, o_2, \ldots, o_T\}$ up to time T, how can the most probable state sequence be determined? Often it is especially interesting to know which is the most probable state at time T. This can be useful to determine the current state of a process. This is known as the *decoding* problem.

3. Given training examples $\mathbf{o}_{\text{train}} = \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_K\}$ of sequences of observations from the process, how can the model parameters $\mathbf{A}, \mathbf{B}, \pi$ be adjusted to maximize $\mathrm{P}(\mathbf{o}_{\text{train}} \mid \lambda)$. This is useful to train the model to match an observed process. This is the problem of *learning*.

Before we go into details on how to solve the three problems, let us consider an example. Assume there are three coins c_1, c_2 and c_3. Only c_1 is a "fair" coin, i.e. has the probability 0.5 for showing head and probability 0.5 for showing tail after a toss of the coin. The coins c_2 and c_3 are rigged so that the probability of showing head is different from showing tail as shown below:

Coin	P(head)	P(tail)
c_1	0.5	0.5
c_2	0.33	0.67
c_3	0.2	0.8

Someone is then asked to toss the fair coin c_1 50 times followed by coin c_2 50 times. With state 1 corresponding to tossing c_1 and state 2 corresponding to tossing c_2 this process can be modeled as a HMM with the following parameters:

$$\begin{aligned}
\mathbf{A} &= \begin{bmatrix} \frac{50}{51} & \frac{1}{51} \\ 0 & 1 \end{bmatrix} \\
\mathbf{B} &= \begin{bmatrix} 0.5 & 0.5 \\ 0.33 & 0.67 \end{bmatrix} \\
\pi &= [1\ 0]
\end{aligned}$$

This can be intuitively explained as follows. It is known that the fair coin is always tossed first, i.e. the probability of starting in state 1 is 1. Since state 1 corresponds to tossing c_1 the probability of head and tail is the same, which can be seen in the $\mathbf{B}$ matrix. Similar for state 2 and c_2. The $\mathbf{A}$ matrix shows the probability of switching from one state to another. State 1 will be active for the first 50 flips, and the 51st flip will induce a switch of state to state 2. Thus the probability of switching to state 2 is 1/51, given that we don't know the number of previous flips which is consistent with the Markov assumption, i.e. the probabilities does not depend on the time. Once in state 2 it will remain active for the rest of the process, and thus it has probability 1 of remaining in state 2.

Now the person is asked to flip c_1 50 times and then to flip *any* of the other two coins for 50 times. The outcome of the flips are recorded and makes up an observation sequence $\mathbf{o}$. From this sequence we are now interested in learning which coins were tossed. To solve this problem using a HMM there are two approaches.

The first approach would be to construct two HMMs, one for the case when c_2 is expected and one for the case when c_3 is expected. For the situation with c_2 the model will obviously be the same as the one above. In the case where c_3 is used the $\mathbf{B}$ matrix must be changed to include the correct probabilities. The second row of the $\mathbf{B}$ would change to [0.2 0.8]. Now there are two models describing the two cases. In order to find out which model is more likely, and thus which coins were used, it is necessary to compute $\mathrm{P}(\mathbf{o} \mid \lambda)$ for both the models and then compare the result. This is an example of the evaluation problem.

The second approach would be to alter the structure of the HMM, introducing a new state. There would then be three states, corresponding to the three coins. The model parameters for such a model would be:

$$
\begin{aligned}
\mathbf{A} &= \begin{bmatrix} \frac{50}{51} & \frac{1}{51\cdot 2} & \frac{1}{51\cdot 2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
\mathbf{B} &= \begin{bmatrix} 0.5 & 0.5 \\ 0.33 & 0.67 \\ 0.2 & 0.8 \end{bmatrix} \\
\pi &= [1\ 0\ 0]
\end{aligned}
$$

In this model, as in the previous model, the probability to switch from state 1 to any other state is $\frac{1}{51}$. The probability to switch to the two other states are distributed evenly since there is no bias towards any of the other coins. To find out which coins were tossed the outcome of the toss is once again recorded in an observation vector $\mathbf{o}$. If now only $\mathrm{P}(\text{state } i \mid \mathbf{o},\ t)$ could be computed for all $t \in [1..100]$ it would be possible to find out which state is most probable at the end of the toss. This requires solving the decoding problem.

So far the examples have been simple an finding the model parameters has been straight forward. In the last example the person is asked to begin tossing any one coin for any number of times followed by another coin for an unknown number of times. To make things even more difficult the person is allowed to use coins where the probability of showing head or tail is unknown. The task now is to determine the number of times the first coin is tossed and the probabilities that each of the two coins show head and tail respectively. To do this it is enough to estimate the model parameters of a two state SLR HMM. This can be done by solving learning problem.

Evaluation

The first problem, the evaluation problem, deals with computing the probability of a sequence of observations symbols $\mathbf{o} = \{o_1, o_2, \ldots, o_T\}$ up to time T, i.e. $\mathrm{P}(\mathbf{o} \mid \lambda)$. Assuming the state sequence traversed is known to be $Q_{\text{known}} = \{q_1, q_2, \ldots, q_T\}$ the probability of the observation sequence can be rewritten as $\mathrm{P}(\mathbf{o}|\lambda) = \mathrm{P}(\mathbf{o}|\lambda, Q_{\text{known}})$. If the state sequence Q traversed is not known it is possible to compute the joint probability of $\mathbf{o}$ and Q as:

$$\mathrm{P}(\mathbf{o}, Q|\lambda) = \mathrm{P}(\mathbf{o}|\lambda, Q)\mathrm{P}(Q|\lambda) \tag{2.1}$$

The probability $\mathrm{P}(\mathbf{o}|\lambda)$ can then be computed by summing over all possible state sequences giving

$$\mathrm{P}(\mathbf{o}|\lambda) = \sum_{\text{all } Q} (\mathrm{P}(\mathbf{o}|\lambda, Q)\mathrm{P}(Q|\lambda)) \tag{2.2}$$

From the model parameters it is now easy to see that

$$\mathrm{P}(\mathbf{o}|\lambda, Q) = \prod_{t=1}^{T} \mathbf{B}_{q_t, \mathbf{o}_t} \tag{2.3}$$

$$\mathrm{P}(Q|\lambda) = \pi_{q_1} \prod_{t=1}^{T-1} \mathbf{A}_{q_t, q_{t+1}} \tag{2.4}$$

One obvious problem with solving the evaluation problem in this way is the summation over all possible state sequences. Since there are N states that can be reached at each time t there will be N^T state sequences. For each such state sequence the equations (2.3) and (2.4) must be computed. This computation involves in the order of $2T$ operations. The complexity of this method is thus in $\mathcal{O}(TN^T)$ which is generally infeasible to compute. As an example, even for the simple model with 2 states and 100 observations, used in the coin toss example, the number of operations would be in the order 10^{32}. Clearly some more efficient approach must be used.

There exists a simple iterative procedure for solving the evaluation problem. It is called the *forward-backward* procedure (Rabiner, 1989; Dugad and Desai, 1996) for reasons that will become obvious. The *forward variable* defined as:

$$\alpha_i(t) = \mathrm{P}(o_1, o_2, \ldots, o_t | q_t = i, \lambda) \tag{2.5}$$

gives the probability of observing the observation sequence $o_1, o_2, \ldots, o_t$ up to time t given that the model at time t is in state i. Realizing that the only way to end up in state $q_t = i$ at time t is to have been in any of the N states at time $t-1$, thus if it was known what the probabilities of being in each state were at time $t-1$, that is $\alpha_i(t-1)$, it would be simple to calculate $\alpha_i(t)$ as:

$$\alpha_i(t) = \mathbf{B}_{i,o_t} \sum_{k=1}^{N} \alpha_k(t-1)\mathbf{A}_{k,i} \tag{2.6}$$

Equation (2.6) states that the probability of being in state i at time t is simply the probability of observing the symbol o_t in that state multiplied with the probability of transferring to this state given the probabilities of each state at time $t-1$. The forward variable can now be computed iteratively using (2.6), given that $\alpha_i(1) = \pi_i \mathbf{B}_{i,o_1}$. To get the total probability of the observation sequence it is enough to sum over the probability of all possible states at time T:

$$\mathrm{P}(\mathbf{o}|\lambda) = \sum_{k=1}^{N} \alpha_k(T) \tag{2.7}$$

The complexity of computing the forward variable is in $\mathcal{O}(N^2T)$, thus the example with the coin toss would require about 400 computations. A drastic reduction from 10^{32}.

The reason that this is referred to as the forward-backward procedure will become evident in the section dealing with the learning problem when the *backward variable* is introduced in a similar way.

Decoding

The decoding Problem deals with computing the most probable state sequence Q_{opt} given a model $\lambda = \{\mathbf{A}, \mathbf{B}, \pi\}$ and a sequence of observations symbols $\mathbf{o} = \{o_1, o_2, \ldots, o_T\}$ up to time T. The problem here is that there is no single definition of optimal. For example, one optimality criterion could be to maximize the expected number of correct individual states. However, the most commonly used criterion is to compute the single best state sequence, i.e. to find Q such that $\mathrm{P}(Q|\mathbf{o}, \lambda)$ is maximized (Rabiner, 1989; Dugad and Desai, 1996). A straightforward approach would be to compute $\mathrm{P}(Q|\mathbf{o}, \lambda)$ for all possible Q. However, as with the evaluation problem the straight forward approach is too computationally expensive. Similar to the forward-backward procedure there exists a famous algorithm to solve the decoding problem, called the Viterbi algorithm (Forney Jr., 1973) which is presented here.

Since $\mathrm{P}(Q|\mathbf{o}, \lambda) = \mathrm{P}(Q, \mathbf{o}|\lambda)/P(\mathbf{o}|\lambda)$ maximizing over $\mathrm{P}(Q, \mathbf{o}|\lambda)$ will result in the same Q, because $P(\mathbf{o}|\lambda)$ is only a constant scaling factor. From (2.1), (2.3) and (2.4) it can be seen that

$$\mathrm{P}(Q, \mathbf{o}|\lambda) = \mathrm{P}(Q|\mathbf{o}, \lambda)P(\mathbf{o}|\lambda) = \prod_{t=1}^{T} \mathbf{B}_{q_t,\mathbf{o}_t} \cdot \pi_{q_1} \prod_{t=1}^{T-1} \mathbf{A}_{q_t,q_{t+1}}$$

Now define

$$\begin{aligned}\Gamma(Q) &= -\ln\left(\mathrm{P}(Q,\mathbf{o}|\lambda)\right) = -\ln\left(\prod_{t=1}^{T}\mathbf{B}_{q_t,\mathbf{o}_t}\cdot\pi_{q_1}\prod_{t=1}^{T-1}\mathbf{A}_{q_t,q_{t+1}}\right) = \\ &= -\left(\ln\left(\pi_{q_1}\mathbf{B}_{q_1,o_1}\right)+\sum_{t=2}^{T}\ln\left(\mathbf{A}_{q_{t-1},q_t}\mathbf{B}_{q_t,o_t}\right)\right)\end{aligned}$$

and note that from this definition

$$\mathrm{P}(Q,\mathbf{o}|\lambda) = e^{-\Gamma(Q)}$$

and thus the problem of maximizing $\mathrm{P}(Q,\mathbf{o}|\lambda)$ becomes equivalent to minimizing $\Gamma(Q)$. This reformulation of the problem is good because it makes it possible to think of terms such as $-\ln(\mathbf{A}_{q_i,q_j}\mathbf{B}_{q_j,o_t})$ as the cost of going from state q_i to state q_j at time t, given that the observation was o_t.

Now that state transitions has been associated with costs it is possible to reformulate the problem as finding the shortest path through a graph. Consider the following: if at time t the shortest route, and its associated cost c_s, to all N states were known it would be possible to compute the shortest route to a state q at time $t+1$ by looking at the cost of going from any of the N states to q and choosing the minimum. Because of the Markov property which states that the next state transition only depends on the current state and the observation at the current time, the shortest path through state q at time t can never change after time t. This means that at any time it is enough to keep track of the shortest path to all N states and from that shortest path at time $t+1$ can be computed. This can then be performed recursively until time T is reached.

To implement the Viterbi algorithm it is necessary to keep track of two properties. First the accumulated cost of being in state i at time t is denoted by $\delta_i(t)$. The second property is the minimum cost of going from state i to state j at time t and is denoted by $\psi_j(t)$. The shortest path can now be computed recursively as

$$\begin{aligned}\delta_j(t) &= \min_i\left(\delta_i(t-1)-\ln(\mathbf{A}_{i,j})\right)-\ln\left(\mathbf{B}_{j,o_t}\right),\ \forall t\in[2,T..] \\ \psi_j(t) &= \operatorname*{argmin}_i\left(\delta_i(t-1)-\ln(\mathbf{A}_{i,j})\right),\ \forall t\in[2,T..]\end{aligned}$$

where, for $i\in[1..N]$

$$\begin{aligned}\delta_i(1) &= -\ln(\pi_i)-\ln(\mathbf{B}_{i,o_t}) \\ \psi_i(1) &= 0\end{aligned}$$

Finally the most probable state at time T, q_T^* and the most probable state sequence leading up to q_T^* is computed as follows

$$
\begin{aligned}
q_T^* &= \underset{i}{\operatorname{argmin}} \left(\delta_i(T)\right) \\
q_t^* &= \psi_{q_{t+1}^*}(t+1), \text{ for } t = T-1, T-2, \ldots, 1
\end{aligned}
$$

The total cost of the optimal path $Q^* = q_1^*, q_2^*, \ldots, q_T^*$ leading to q_T^* is thus $P^* = \min_i \left(\delta_i(T)\right)$ and the associated probability is given by e^{-P^*}.

Learning

The problem of determining the model parameters $\mathbf{A}, \mathbf{B}, \pi$ of the model λ in order to maximize the probability $\mathrm{P}(\mathbf{o} \mid \lambda)$ is by far the most difficult of the three problems. This problem is somewhat different than the other two in that there exists no analytical solution. As a matter of fact there is no optimal way of estimating the model parameters given a finite observation sequence (Rabiner, 1989; Dugad and Desai, 1996). However, the model parameters can be locally optimized. One famous method for accomplishing this is the Baum-Welch method which is described here. The model parameters that needs to be estimated are $\mathbf{A}, \mathbf{B}$ and π. With the same reasoning as in the coin flip example used previously a natural way of estimating them would be

$$
\begin{aligned}
\pi_i^* &= \text{expected number of times in state } i \text{ at time } t = 1 && (2.8) \\
\mathbf{A}_{i,j}^* &= \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} && (2.9) \\
\mathbf{B}_{i,o}^* &= \frac{\text{expected number of times in state } i \text{ and observing } o}{\text{expected number of times in state } i} && (2.10)
\end{aligned}
$$

Now all that has to be done is to compute these properties from the training sequences. In order to achieve this define

$$
\xi_{i,j}(t) = \mathrm{P}(q_t = i, q_{t+1} = j | \mathbf{o}, \lambda) \tag{2.11}
$$

as the probability of being in state i at time t followed by state j at time $t+1$ given the model λ and the observation sequence $\mathbf{o}$. To compute ξ it is now time to introduce the backward variable of the forward-backward procedure mention previously while solving the evaluation problem. Recall that the forward variable $\alpha_i(t)$ gives the probability of being in state i at time t given the observations sequence $\mathbf{o} = \{o_1, o_2, \ldots, o_t\}$ up to time t for a given model λ. The backward variable $\beta_i(t)$ gives the probability of the observation sequence $o_{t+1}, o_{t+2}, \ldots, o_T$ given the model λ and that the state at time t is i, that is $\beta_i(t) = \mathrm{P}(o_{t+1}, o_{t+2}, \ldots, o_T | q_t = i, \lambda)$. The backward variable can be computed in a similar manner to the forward variable:

$$
\begin{aligned}
\beta_i(T) &= 1, \ \forall\ i \in [1..N] \\
\beta_i(t-1) &= \sum_{j=1}^{N} \left(\mathbf{A}_{i,j} \mathbf{B}_{j,o_t} \beta_j(t)\right)
\end{aligned}
$$

The forward and backward variables can now be used in conjunction with Bayes rule to compute (2.11) in the following way:

$$\begin{aligned}
\xi_{i,j}(t) &= \mathrm{P}(q_t = i, q_{t+1} = j|\mathbf{o}, \lambda) = \\
&= \frac{\mathrm{P}(q_t = i, q_{t+1} = j, \mathbf{o}|\lambda)}{\mathrm{P}(\mathbf{o}|\lambda)} = \\
&= \frac{\alpha_i(t)\mathbf{A}_{i,j}\mathbf{B}_{j,o_{t+1}}\beta_j(t+1)}{P(\mathbf{o}|\lambda)} = \\
&= \frac{\alpha_i(t)\mathbf{A}_{i,j}\mathbf{B}_{j,o_{t+1}}\beta_j(t+1)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i(t)\mathbf{A}_{i,j}\mathbf{B}_{j,o_{t+1}}\beta_j(t+1)} \qquad (2.12)
\end{aligned}$$

To solve the learning problem it is only required to introduce one more property, $\gamma_i(t) = \mathrm{P}(q_t = i|\mathbf{o}, \lambda)$, which is the probability of being in state i at time t given the model λ and the observation sequence $\mathbf{o}$ up to time t. Using Bayes rule and the forward and backward variables $\gamma_i(t)$ can be written as:

$$\gamma_i(t) = \mathrm{P}(q_t = i|\mathbf{o}, \lambda) = \frac{\mathrm{P}(q_t = i, \mathbf{o}|\lambda)}{\mathrm{P}(\mathbf{o}|\lambda)} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N}\alpha_j(t)\beta_j(t)}$$

and it relates to ξ as

$$\gamma_i(t) = \sum_{j=1}^{N}\xi_{i,j}(t)$$

Since $\gamma_i(t)$ is the probability for being in state i at time t summing $\gamma_i(t)$ over all $t \in [1..T]$ gives the expected number of times state i is visited. Summing only up until $t = T - 1$ yields the expected number of transitions out of state i, that is:

$$\begin{aligned}
\rho_i &= \sum_{t=1}^{T}\gamma_i(t) && = \text{expected number of times in state } i \\
\sigma_i &= \sum_{t=1}^{T-1}\gamma_i(t) && = \text{expected number of transitions from state } i
\end{aligned}$$

Similarly summing $\xi_{i,j}(t)$ over all $t \in [1..T-1]$ gives the expected number of transitions from state i to state j.

$$\upsilon_{i,j} = \sum_{t=1}^{T-1}\xi_{i,j}(t) = \text{expected number of transitions from state } i \text{ to state } j$$

Now all properties that is necessary to update λ can be computed as:

$$\begin{aligned} \pi_i^* &= \gamma_i(1) \\ \mathbf{A}_{i,j}^* &= \frac{\upsilon_{i,j}}{\sigma_i} \\ \mathbf{B}_{i,k}^* &= \frac{\sum_{t=1}^{T} \begin{cases} 0, & o_t \neq O_k \\ \gamma_i(t), & o_t = O_k \end{cases}}{\rho_i} \end{aligned} \tag{2.13}$$

where $\mathbf{B}_{i,k}^*$ is simply the number of times in state i while observing O_k.

The problem with this is that in order to compute the optimal values from the forward and backward variables it is necessary to know the model parameters, thus to compute the model parameters the model parameters have to be known. While this might seem like a catch-22 there is a simple solution to the problem. The solution is to start with an initial guess of the parameters and keep reestimating them using the equations above until a local maximum is found. Starting with an initial model $\lambda(\mathbf{A}, \mathbf{B}, \pi)$ and using the parameters of that model to compute a new model $\lambda^*(\mathbf{A}^*, \mathbf{B}^*, \pi^*)$ it can be shown that either $\lambda^* = \lambda$ or $\mathrm{P}(\mathbf{o}|\lambda^*) > \mathrm{P}(\mathbf{o}|\lambda)$. Now it is simply a matter of using the new model λ^* as an initial guess and keep reestimating the parameters iteratively until $\lambda^* = \lambda$, in which case a local maximum has been found.

While the Baum-Welch method is the most famous algorithm for training HMMs there are other methods. One other method is the segmental k-means algorithm (Dugad and Desai, 1996; Juang and Rabiner, 1990). The Baum-Welch algorithm adapts λ in order to locally maximize $\mathrm{P}(\mathbf{o}|\lambda)$. The segmental k-means algorithm adapts λ in order to locally maximize $\mathrm{P}(\mathbf{o}, Q|\lambda)$. As the name implies the segmental k-means algorithm is based on k-means clustering (see section 2.1.5).

Since the Baum-Welch method is a local method that will search for a local maximum of the probability $\mathrm{P}(\mathbf{o}|\lambda)$ the initialization of λ can be very important. Not only will the choice of the initial parameters affect the rate of convergence of the Baum-Welch algorithm but it will also determining which local maximum is found. Furthermore the initialization can be used to constrain the model in various ways. For example from (2.12) it is clear that if $\mathbf{A}_{i,j}$ is zero $\xi_{i,j}(t)$ will be zero for all t. This results in that $\upsilon_{i,j}$ will be zero in (2.14) and thus $\mathbf{A}_{i,j}$ will remain zero for ever. This can be very useful to constrain the model. As an example, consider creating a SLR model. It would then be reasonable to initialize $\mathbf{A}$ as $\mathbf{A}_{i,j} = 0,\ \forall\ i \neq j,\ i \neq j - 1$ and $\mathbf{A}_{i,j} = 1 - \epsilon,\ \forall\ i = j$ and $\mathbf{A}_{i,j} = \epsilon,\ \forall\ i = j - 1$. The value ϵ can be chosen arbitrarily in the interval $]0, 1[$ but will affect the rate of convergence and possibly which maxima is found. In this way the model can be initialized to allow, for example, stepping backwards or even to be fully connected.

Extensions to HMMs

Hidden Markov models have been applied successfully in several areas and a number of extensions has been proposed. Some of these extensions that are relevant to the work presented in this thesis is briefly described in the following sections. The interested reader are referred to the references for further details.

Multi dimensional HMM

In some settings it is not convenient to map multi dimensional input to a set of enumerated symbols. Hannaford and Lee (1990) suggests a different approach. The idea is to extend the HMM classification to directly deal with multi dimensional observation symbols. The Viterbi algorithm uses the probability $P(o_t|\text{state } i)$ of observing a specific observation symbol o_t in state i together with the state transition probabilities to compute the probability of being in a certain state. To be able to extend the Viterbi algorithm to multidimensional data, independence between the different dimensions is assumed. Thus there will be a $\mathbf{B}$ matrix for each dimension of the input data. This means that for a D dimensional HMM the observation symbols are also D dimensional where each dimension d contains values from a finite enumerated set $\mathcal{O}_d = \{O_1, O_2, \ldots, O_{K_d}\}$. The recursion equation in the forward procedure (2.6) now becomes

$$\alpha_i(t) = \left(\sum_{k=1}^{N} \alpha_k(t-1)\mathbf{A}_{k,i}\right) \prod_{d=1}^{D} \mathbf{B}_{i,o_{d,t}}$$

Continuous Density HMM

Another limitation with the basic HMM formulation is that the observations must be taken from a finite enumerated set $\mathcal{O} = \{O_1, O_2, \ldots, O_M\}$. This can often be achieved by clustering or quantization of the data, (Gray, 1984) but sometimes this is not a good approach. To overcome this the Continuous Density HMM, or CDHMM, has been proposed. Here the $\mathbf{B}$ matrix is replaced by a set of probability density functions (PDFs) that are used to estimate the required $P(o|\text{state } i)$. Usually a Gaussian PDF or a mixture of Gaussians is used, but any PDF can be chosen to fit a particular application. In the case of a Gaussian PDF the probability of observing an observation o given a state i can be computed as

$$P(o|\text{state } i) = \frac{1}{\sqrt{2\pi\sigma_i}} e^{\frac{(o-m_i)^2}{2\sigma_i^2}} \tag{2.14}$$

where m_i is the mean and σ_i is the standard deviation of the Gaussian. If a mixture of K Gaussians is used there will be an additional scaling factor such that (2.14) becomes

$$P(o|\text{state } i) = \sum_{k=1}^{K} \frac{c_{i,k}}{\sqrt{2\pi\sigma_i}} e^{\frac{(o-m_i)^2}{2\sigma_i^2}} \tag{2.15}$$

All the parameters m_i, σ_i and $c_{i,k}$ can be estimated by straightforward modifications to the Baum-Welch algorithm, see e.g. Gauvain and Chin-Hui (1994).

Probability Estimators for HMMs

Using a parametric distribution to model the observation probabilities as mentioned above may decrease the performance of the HMM since the real distribution is hidden and the assumption of a parametric distribution is a strong hypothesis on the model (Castellani *et al.*, 2004).

Applying probability estimators avoids this problem. These estimators compute the observation symbol probability (Bourlard and Morgans, 1990) instead of using a look-up matrix or parametric model. Another advantage with probability estimators compared to using discrete symbols is the possibility to use continuous input instead of discrete observation symbols for the HMM.

Successful use of probability estimators using multi layer perceptions (MLP) and Support Vector Machines (SVMs) are reported in (Bourlard and Morgans, 1990; Renals *et al.*, 1994; Ganapathiraju *et al.*, 2000). Naturally, the estimators cannot be trained with the Baum-Welch algorithm, but have to be trained separately.

Optimization of the HMM Parameters

It has been shown in section 2.1.2 how the parameters $\mathbf{A}, \mathbf{B}, \pi$ of a HMM can be trained. However, there still exists a number of free parameters for the HMM, such as the number of states N, the topology of the model, the number of observation symbols M or Gaussians etc. No general solution on how to estimate all of the above mentioned parameters from training data is available (Zimmermann and Bunke, 2002). However, methods have been proposed for optimizing the number of states, the number of Gaussians and the number of training iterations, (Günter and Bunke, 2003; Zimmermann and Bunke, 2002).

2.1.3 Structured Hidden Markov Models

It is sometimes useful to use HMMs in specific structures in order to facilitate learning and generalization. For example, even though a fully connected HMM could always be used if enough training data is available it is often useful to constrain the model by not allowing arbitrary state transitions. In the same way it can be beneficial to embed the HMM into a greater structure; which, theoretically, may not be able to solve any other problems than the basic HMM but can solve some problems more efficiently when it comes to the amount of required training data.

Layered Hidden Markov Model

A layered hidden Markov model (LHMM), (Oliver *et al.*, 2004) consists of N levels of HMMs where the HMMs on level $N+1$ corresponds to observation symbols or probability generators at level N. Every level i of the LHMM consists of K_i HMMs running in parallel, see figure 2.2.

At any given level L in the LHMM a sequence of T_L observation symbols $\mathbf{o}_L = \{o_1, o_2, ..., o_{T_L}\}$ can be used to classify the input into one of K_L classes, where each class corresponds to each of the K_L HMMs at level L. This classification can then be used to generate a new observation for the level $L-1$ HMMs. At the

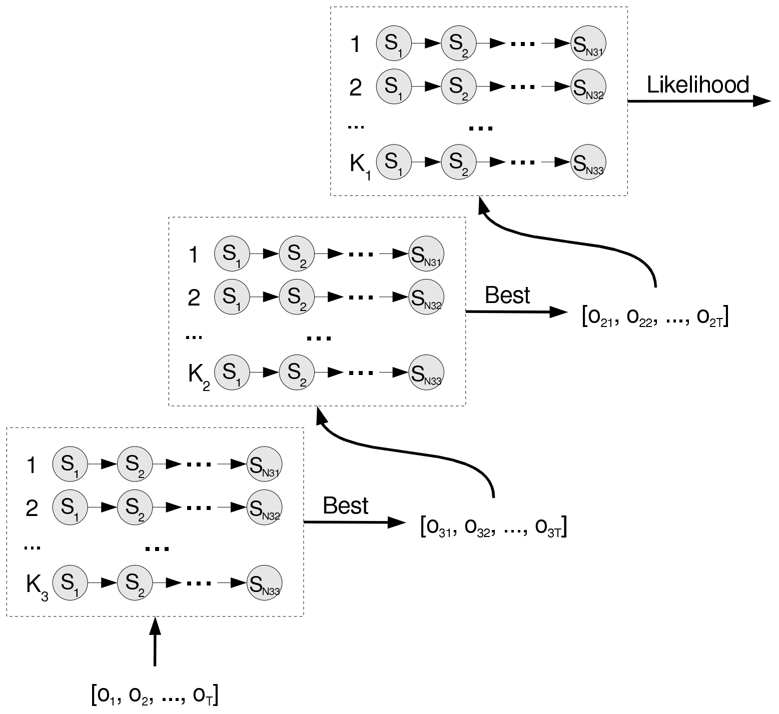


Figure 2.2. A layered hidden Markov model

lowest layer, i.e. level N, primitive observation symbols $\mathbf{o}_p = \{o_1, o_2, ..., o_{T_p}\}$ would be generated directly from observations of the modeled process. For example in a trajectory tracking task the primitive observation symbols would originate from the quantized sensor values. Thus at each layer in the LHMM the observations originate from the classification of the underlying layer, except for the lowest layer where the observation symbols originate from measurements of the observed process.

It should be noted here that it is not necessary to run all levels at the same time granularity. For example it is possible to use windowing at any level in the structure so that the classification takes the average of several classifications into consideration before passing the results up the layers of the LHMM.

Instead of simply using the winning HMM at level $L+1$ as an input symbol for the HMM at level L it is possible to use it as a probability generator by passing the complete probability distribution up the layers of the LHMM. Thus instead of having a "winner takes all" strategy where the most probable HMM is selected as an observation symbol, the likelihood $L(i)$ of observing the ith HMM can be used in the recursion formula of the level L HMM to account for the uncertainty in the classification of the HMMs at level $L+1$. Thus, if the classification of the HMMs at level $n+1$ is uncertain, it is possible to pay more attention to the a-priori information encoded in the HMM at level L.

It should be noted here that a LHMM could in practice be transformed into a single layered HMM where all the different models are concatenated together. Some of the advantages that may be expected from using the LHMM over a large single layer HMM is that the LHMM is less likely to suffer from over-fitting since the individual sub-components are trained independently on smaller amounts of data. A consequence of this is that a significantly smaller amount of training data is required for the LHMM to achieve a performance comparable of the HMM. Another advantage is that the layers at the bottom of the LHMM, which are more sensitive to changes in the environment such as the type of sensors, sampling rate etc, can be retrained separately without altering the higher layers of the LHMM.

Hierarchical Hidden Markov Models

In the Hierarchical Hidden Markov Model (HHMM) each state is considered to be a self contained probabilistic model. More precisely each state of the HHMM is itself a HHMM. This implies that the states of the HHMM emits sequences of observation symbols rather then single observation symbols as is the case for the standard HMM states.

When a state in a HHMM is activate, it will activate its own probabilistic model, i.e. it will activate one of the states of the underlying HHMM, which in turn may activate its underlying HHMM and so on. The process is repeated until a special state, called a production state, is activated. Only the production states emit observation symbols in the usual HMM sense. When the production state has emitted a symbol, control returns to the state that activated the production state. The states that does not directly emit observations symbols are called internal states. The activation of a state in an HHMM under an internal state is called a *vertical transition*. After a vertical transition is completed a *horizontal transition* occurs to a state within the same level. When a horizontal transition leads to a *terminating* state control is returned to the state in the HHMM, higher up in the hierarchy, that produced the last vertical transition.

Remember that a vertical transition can result in more vertical transitions before reaching a sequence of production states and finally returning to the top level. Thus the production states visited gives rise to a sequence of observation symbols that is “produced” by the state at the top level. The structure of the HHMM is shown in figure 2.3.

The methods for estimating the HHMM parameters and model structure are more complex than for the HMM and the interested reader is referred to (Fine *et al.*, 1998).

It should be pointed out that the HMM, HHMM and LHMM all belong to the same class of classifiers. That is, they can be used to solve the same set of problems. In fact, both the HHMM and LHMM can be transformed into a standard HMM. However, both HHMMs and LHMMs utilize their structure to solve a subset of the problems more efficiently.

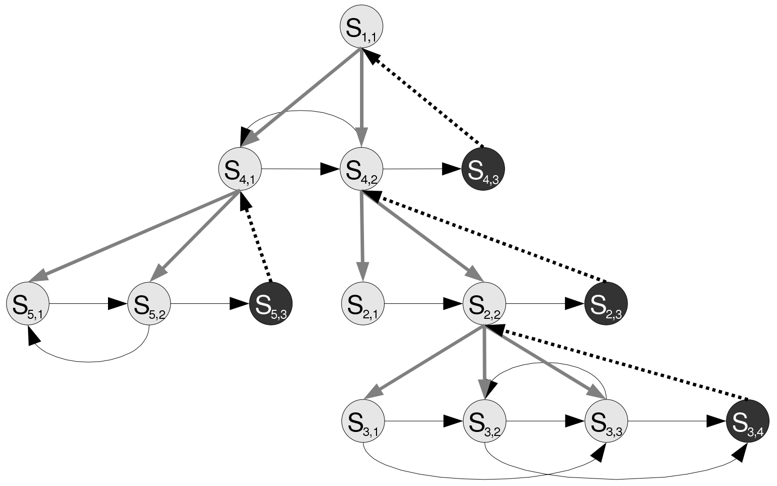


Figure 2.3. Illustration of the structure of a HHMM. Gray lines shows vertical transitions. The horizontal transitions are shown as black lines. The light gray circles are the internal states and the dark gray circles are the terminal states that returns control to the activating state. The production states are not shown in this figure.

2.1.4 Support Vector Machines

Support Vector Machines (SVMs) have been used extensively for pattern classification in a number of research areas (Roobaert, 2001; Rychetsky *et al.*, 1999; Hyunsoo and Haesun, 2004). SVMs have several appealing properties such as fast training, accurate classification and good generalization (Chen *et al.*, 2005; Burges, 1998). In short, SVMs are binary classifiers that separate two classes by an optimal separation hyperplane. The separation hyperplane is found by minimizing the expected classification error which is equal to maximizing the margin as demonstrated in figure 2.4.

SVMs work with linear separation surfaces in a pre-Hilbert space (Chen *et al.*, 2005), i.e. a space where the inner-product is defined. However, the input patterns are often not linearly separable, or even defined in such a space. To overcome this limitation, a “kernel trick” is used to transform the input pattern to a Hilbert space (Aizerman *et al.*, 1964). A map $\phi : \mathcal{X} \rightarrow \mathcal{H}$ is defined for the patterns $x \in \mathcal{X}$. The Hilbert space $\mathcal{H}$ is commonly called the feature space (Chen *et al.*, 2005). There are three benefits of transforming the data into $\mathcal{H}$ (Chen *et al.*, 2005). It makes it possible to define a similarity measure from the dot product in $\mathcal{H}$. In addition, it provides a setting to deal with the patterns geometrically and moreover makes it possible to study learning algorithms using linear algebra and analytic geometry. Finally, it provides the freedom to choose the mapping ϕ which, in its turn, makes it possible to design a large variety of learning algorithms. SVMs try to estimate a function $h : \mathcal{X} \rightarrow \{\pm 1\}$ that classifies the input $x \in \mathcal{X}$ to one of

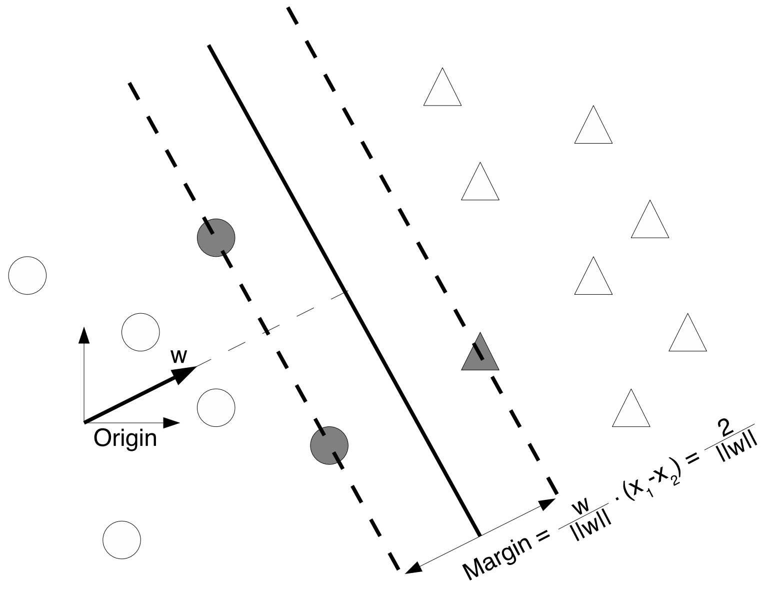

w
Origin
Margin = $\frac{w}{\|w\|} \cdot (x_1 - x_2) = \frac{2}{\|w\|}$

Figure 2.4. A binary classification example: circles are separated from triangles by a separation hyperplane. The support vectors originate from the training examples and are marked by filled symbols.

the two classes ±1 based on input-output training data. It is a well known fact from Vapnik-Chervonenkis (VC) theory that it is imperative to restrict the class of functions that h is chosen from, in order to avoid over-fitting. SVMs restrict the set of functions to hyperplanes.

Consider the class of hyperplanes

$$\mathbf{w} \cdot \mathbf{x} + b = 0,\ \mathbf{w}c \in \mathbb{R}^N,\ b \in \mathbb{R}$$

with $\mathbf{x} = \phi(x)$ and the corresponding decision function

$$h(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Among all such hyperplanes there exists a unique one that gives the maximum margin of separation between the two classes ±1 (Chen *et al.*, 2005), that is:

$$\max_{\mathbf{w},b} \left(\min(\|\mathbf{x} - \mathbf{x}_i\| : \mathbf{x} \in \mathbb{R}^N,\ \mathbf{w} \cdot \mathbf{x} + b = 0) \right) \tag{2.16}$$

If $\mathbf{w}$ and b are scaled to obtain a canonical form, i.e. $\mathbf{w} \cdot \mathbf{x}_1 + b = 1$ and $\mathbf{w} \cdot \mathbf{x}_2 + b = -1$, the size of the margin becomes $2/\|\mathbf{w}\|$. The optimal hyperplane can then be computed by solving the following optimization problem:

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2 \text{ over } \mathbf{w}, b \\ &\text{subject to : } y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1,\ \forall i \in [1..M] \end{aligned} \tag{2.17}$$

One way to solve (2.17) is through the Lagrangian dual:

$$\max_{\alpha \geq 0} \left(\min_{\mathbf{w},b} \left(L(\mathbf{w}, b, \alpha) \right) \right)$$

$$L(\mathbf{w}, b, \alpha) = \quad \frac{2}{\|\mathbf{w}\|^2} - \sum_{i=1}^{M} \alpha_i (y_i((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1)$$

From the above, it can be shown (Chen *et al.*, 2005) that the hyperplane decision function can be written as:

$$h(\mathbf{x}) = \text{sgn} \left(\sum_{i=1}^{m} y_i \cdot \alpha_i (\mathbf{x} \cdot \mathbf{x}) + b \right) \tag{2.18}$$

which implies that the solution vector $\mathbf{w}$ has an expansion in terms of a subset of the training samples. The subset is formed by the training samples with a non-zero Lagrange multiplier, α_i. The samples with a non-zero Lagrange multiplier are known as the *support vectors*. The support vectors can easily be computed by solving a quadratic programming problem, (Chen *et al.*, 2005).

The basic SVM formulation achieves poor classification performance when faced with outliers. That is if it is impossible to clearly separate the two classes with a separation hyperplane. This can happen if the noise is high so that there is an overlap between the two classes. There exists modifications to the basic SVM formulation so that this can be handled. For example it is possible to introduce an additional variable ν known as the slack variable. This variable can be used to allow some amount of the training examples to violate (2.17), effectively creating a "soft" margin where not all training examples need to be on the correct "side" of the separation hyperplane, see e.g. Chen *et al.* (2005).

2.1.5 K-Means Clustering

K-means clustering is a procedure for partitioning an N-dimensional population into k sets given training data extracted from that population, (MacQueen, 1967; Jain *et al.*, 1999). The k-means procedure is both computationally efficient and easy to implent, while maintaining good classification performance. Some variants of the k-means algorithm exit, but the core idea is the same. The approach used in this thesis is given here, see (MacQueen, 1967) for details and analysis of the algorithm.

The core idea of the k-means algorithm is to find k cluster centers $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$ given a set of training data $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ so that $\mathbf{x}_i$ is said to be a *member* of cluster $\mathbf{c}_j$ if, according to some distance measure $\delta(\cdot, \cdot)$:

$$\delta(\mathbf{x}_i, \mathbf{c}_j) < \delta(\mathbf{x}_i, \mathbf{c}_l) \; \forall j, l \in [1..k], \; j \neq l$$

In the case where $\delta(\mathbf{x}_i, \mathbf{c}_j) = \delta(\mathbf{x}_i, \mathbf{c}_l), \; j < l, \; \mathbf{x}_i$ is a member of $\mathbf{c}_j$ so that if $\mathbf{c}_j = \mathbf{c}_l$ and $j < l$ $\mathbf{c}_l = \emptyset$. That is, each datum is a member of the cluster corresponding to its closes cluster center and if two or more cluster centers are at the same distance it is a member of the lowest numbered cluster center. The set of data points $\hat{X} \subseteq X$ that is a member of cluster $\mathbf{c}$ is denoted by $\hat{X} = \mu(X, \mathbf{c})$. The algorithm used to find the cluster centers can now be described by algorithm 1.

Algorithm 1 K-means clustering

```
1. Initialize $\mathbf{C} = [\mathbf{c}_1; \mathbf{c}_2; \ldots; \mathbf{c}_k]$ to $k$ random data points from the training data $X$.
2. Find the members of each data point so that $\mathbf{S}_i$ contains all members to cluster $i$
3. Compute a new cluster center $\mathbf{c}_i$ from the average of all members in $\mathbf{S}_i$.
4. If the cluster centers are stationary, return $\mathbf{C}$
5. Generate a new $\mathbf{C}$ from the new cluster centers and repeat from 2
```

The number of clusters and the initial cluster centers will affect the performance and the result of the algorithm, (Jain *et al.*, 1999). To minimize the risk of a poor initial selection of clusters the algorithm can be run several times to generate different clusters and the result with, for example, the lowest average or total distance of data points to their corresponding cluster center can be chosen.

A way to adapt the number of clusters was proposed by MacQueen (1967). The idea is to introduce two new variables C and R known as the *coarsening* and *refinement* parameters. After each iteration of the k-means algorithm the minimum distance between two cluster centers are determined, such that

$$\Delta = \min_{i,j} \left(\delta(\mathbf{c}_i, \mathbf{c}_j) \right) \tag{2.19}$$

if $\Delta < C$ the two corresponding clusters are merged using their weights. In this thesis the weights are simply the number of training points assigned to each cluster center. This results in a coarsening of the clusters. Similarly, for the refinement process, the distance of each data point to its corresponding cluster, $\delta(\mathbf{x}_j, \mathbf{c}_i)$, is compared to R so that if $\delta(\mathbf{x}_j, \mathbf{c}_i) > R$, $\mathbf{x}_j$ is taken to be a new cluster center. This result is in a refinement of the clusters. One problem with this approach is that the selection of C and R can be difficult. For a known domain C and R can often be given intuitive values. For a domain where little is known about the data the selection of the parameters becomes more difficult. Furthermore, the parameters are sensitive to scaling of the data.

Another approach that can be applied to many clustering algorithms is the so called *elbow criterion.* The basic idea is simple. New clusters are added as long as they provide a sufficient increase in performance. Now all that is needed is a definition of sufficient and performance. To understand the elbow criterion consider

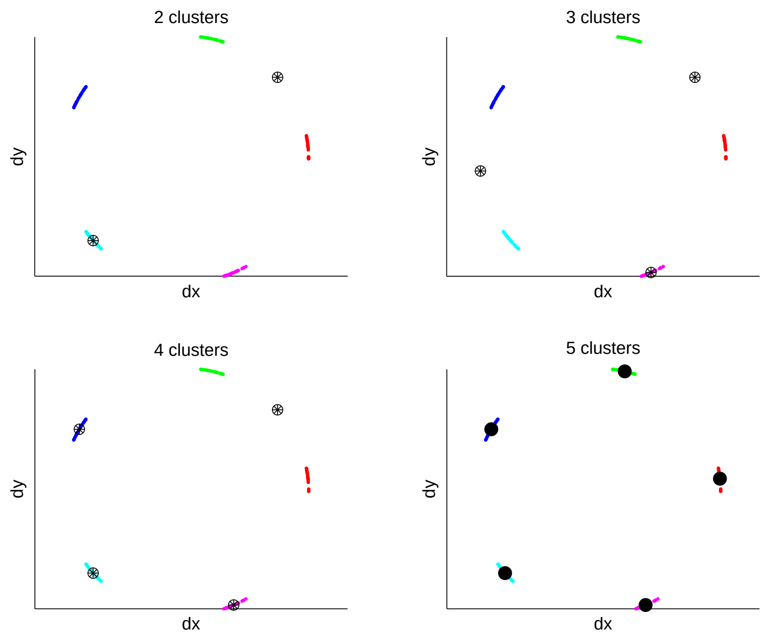


Figure 2.5. Cluster centers obtained by k-means clustering of motion direction vectors for five nominal directions.

the following example. The goal is to estimate the number of motion directions in a data set. The input exemplars, i.e. the training set, is a set of motion vectors. These vectors lie on the unit circle as shown in figure 2.5 for 2, 3, 4 and 5 clusters. In this case a reasonable measure of the performance of the clustering using k clusters is based on the sum of distances from all exemplars to their corresponding cluster center as shown in (2.20). This performance measurement is by no means unique but it serves to illustrate the point.

$$\upsilon(k) = \sum_{i=1}^{k} \sum_{\forall \mathbf{x} \in \mu(X, \mathbf{c}_j)} \delta(\mathbf{x}_i, \mathbf{c}_j) \tag{2.20}$$

Next $\upsilon(k)$ is computed for a range of possible values of k, $k \in [1..15]$ in this example. It is now possible to compute the normalized error $e(k) \in [0,1]$ from $\upsilon(k)$ as

$$e(k) = \frac{\upsilon(k) - \min_i \upsilon(i)}{\max_j \upsilon(j)}$$

The error $e(k)$ can now be used as a performance measure so that when the error is low, the performance is high, and vice versa. Figure 2.6 (top) shows a plot of

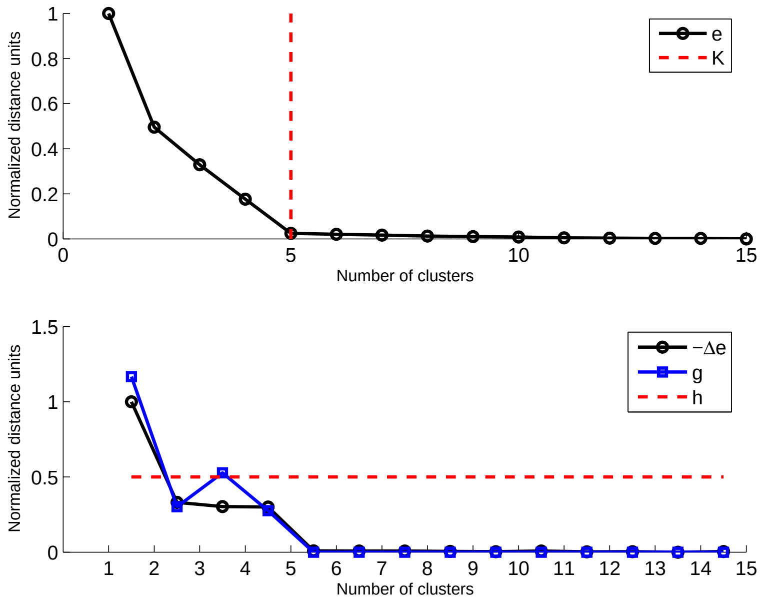


Figure 2.6. Estimation of the number of clusters using the elbow criterion. The actual number of clusters is 5.

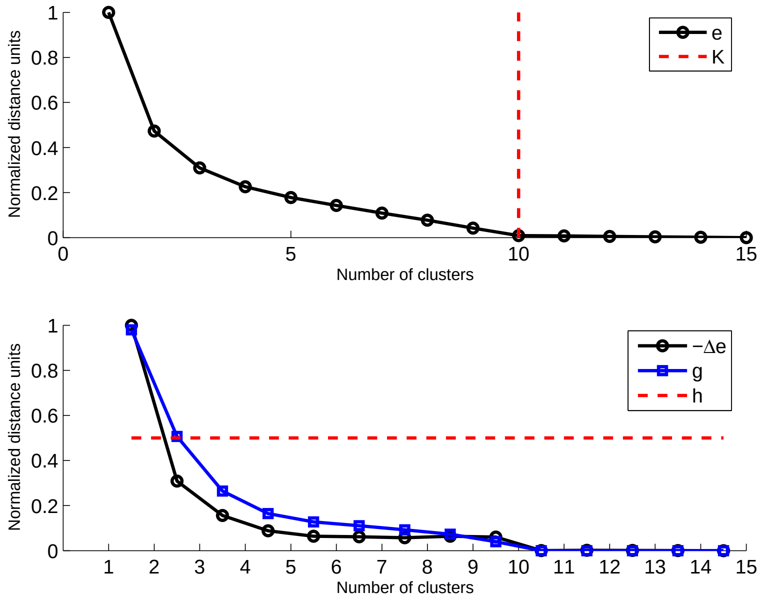


Figure 2.7. Estimation of the number of clusters using the elbow criterion. The actual number of clusters is 10.

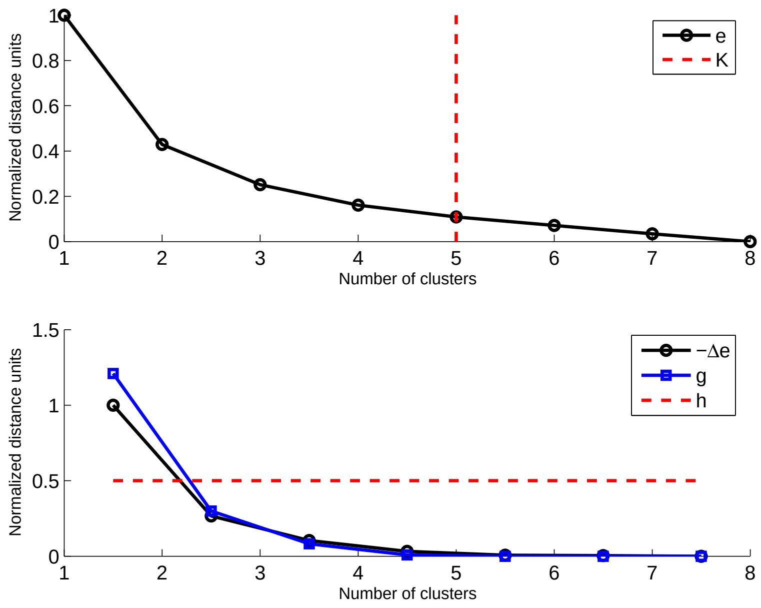


Figure 2.8. Estimation of the number of clusters using the elbow criterion. The actual number of clusters is 10, but only 8 clusters are considered and therefore the wrong elbow point is selected.

$e(k)$ for the example data set shown in figure 2.5. It can be seen from figure 2.6 (top) that the rate of decrease in error drops sharply around 5 clusters, the curve has an "elbow" at $k = 5$. The idea is that when the addition of more clusters does not have a significant effect on the error compared to the previous clusters added the optimal number of clusters has been found. The lower plot of figure 2.6 shows the slope $\Delta e(k)$ of $e(k)$. As it can be seen the slope is close to zero for $k > 5$. By manually inspecting the normalized error or its slope it is easy to find the number of clusters. However, if no manual intervention is allowed it is necessary to automatically estimate the "elbow" point. The simplest way to do this is to put a threshold on $e(k)$ or $-\Delta e(k)$ so that when the value drops below this threshold the correct number of clusters has been found. A more robust approach is to apply cumulative sum (CUSUM) RLS test (Gustafsson, 2000) to Δe. The test statistic $g(k)$ of the CUSUM test for the input $y(k)$ is updated according to (2.21).

$$
\begin{aligned}
& g(n+1) = g(n) + y(n) - \hat{y}(n) - \nu && (2.21)\\
& g(n) = 0, \text{ and } k^* = n \text{ if } g(n) < 0 && (2.22)\\
& g(n) = 0, \; n_a = n \text{ and alarm if } g(n) > h \\
& \hat{y}(n+1) = \hat{y}(t) \cdot \frac{N-1}{N} + \frac{y(n)}{N}, \; N = n - n_0
\end{aligned}
$$

Running the CUSUM RLS test backwards from $n = K$ yields the test statistic shown in figure 2.6 (bottom). The test triggers when $g(k) > h$, which is when the dashed line is crossed. The change is estimated to the last sample the test was reset, i.e. $g(k) < 0$, this point marks the elbow of the normalized error and is shown as a vertical dashed line in figure 2.6.

Figure 2.7 shows the same thing for an example with 10 clusters. From the figure it can be seen that there are actually "two" elbows, one around $k = 5$, and one more distinct around $k = 10$. This is natural because at $k = 5$ each cluster center is assigned to two actual clusters, drastically improving the average error compared to using fewer clusters. This can be a problem if the range of values for k investigated is too small. For example, figure 2.8 shows an example with the same data, i.e. 10 clusters, but where only the possibility of 1-8 clusters are examined. It can be seen that in this case the test triggers and gives an estimate of the number of clusters of 5, which is the "strongest" elbow in this range.

2.2 Virtual Fixtures

Rosenberg (1993) presents the concept of *virtual fixtures* as an overlay of abstract sensory information on a workspace in order to improve the tele-presence in a tele-manipulation task. The concept of abstract sensory overlays is difficult to visualize and talk about, as a consequence the virtual fixture metaphor was introduced. To understand what a virtual fixture is an analogy with a real physical fixture such as a ruler is often used. A simple task such as drawing a straight line on a piece of paper on free-hand is a task that most humans are unable to perform with good accuracy and high speed. However, the use of a simple device such as a ruler allows the task to be carried out fast and with good accuracy. The use of a ruler helps the user by guiding the pen along the ruler reducing the tremor and mental load of the user, thus increasing the quality of the task.

The definition of virtual fixtures in (Rosenberg, 1993) is much broader than simply providing guidance of the end-effector. For example, auditory virtual fixtures are used to increase the user awareness by providing audio clues that helps the user by providing multi modal cues for localization of the end-effector. Rosenberg argues that the success of virtual fixtures is not only due to the fact that the user is guided by the fixture, but that the user experiences a greater presence and better localization in the remote workspace. However, in the context of HMCS, the term virtual fixture is most often used to refer to a task dependent aid that guides the user's motion along desired directions while preventing motion in undesired directions or regions of the workspace. This is the definition of virtual fixture used in this thesis; if not stated otherwise.

Virtual fixtures can be either *guiding virtual fixtures* or *forbidden regions virtual fixtures*. A forbidden regions virtual fixture could be used, for example, in a tele-operated setting where the operator has to drive a vehicle at a remote site to accomplish an objective. If there are pits at the remote site which would be harmful

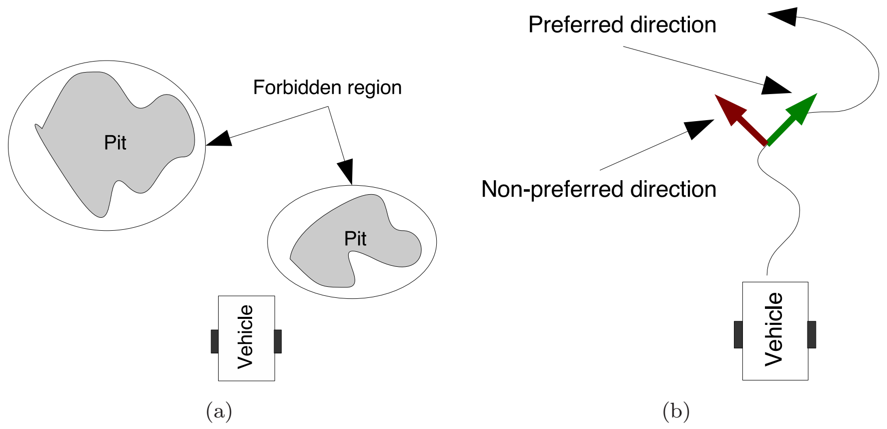


Figure 2.9. Example of virtual fixtures. (a) Forbidden regions virtual fixtures. (b) Guiding virtual fixtures.

for the vehicle to fall into; forbidden regions could be defined at the various pits locations, thus preventing the operator from issuing commands that would result in the vehicle ending up in such a pit, see figure 2.9(a). Such illegal command could easily be sent by an operator because of, for instance, delays in the tele-operation loop, bad tele-presence or a number of other reasons.

An example of a guiding virtual fixture could be when the vehicle must follow a certain trajectory, see figure 2.9(b).

The operator is then able to control the progress along the *preferred direction* while motion along the *non-preferred direction* is constrained.

With both forbidden regions and guiding virtual fixtures the *stiffness*, or its inverse the *compliance*, of the fixture can be adjusted. If the compliance is high (low stiffness) the fixture is *soft*. On the other hand when the compliance is zero (maximum stiffness) the fixture is *hard*, see figure 2.10. Recent research has inves-

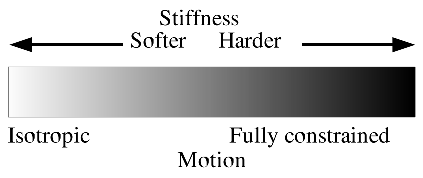


Figure 2.10. The stiffness of a virtual fixture can be soft or hard. A hard fixture completely constrains the motion to the fixture while a softer fixture allows some deviations from the fixture.

tigated different ways of adjusting the compliance of the virtual fixture to further improve the execution of collaborative tasks, (Nolin *et al.*, 2003; Marayong *et al.*, 2002).

2.2.1 Virtual Fixture Control Law

This section describes how a control law that implements virtual fixtures can be derived. It is assumed that the robot is a purely kinematic device with end-effector position $\mathbf{p} = [x, y, z] \in \mathbb{R}^3$ and end-effector orientation $\mathbf{r} = [r_\mathrm{x}, r_\mathrm{y}, r_\mathrm{z}] \in \mathbb{R}^3$ expressed in the robot's base frame F_r. The input control signal $\mathbf{u}$ to the robot is assumed to be a desired end-effector velocity $\mathbf{v} = \dot{\mathbf{x}} = [\dot{\mathbf{p}}, \dot{\mathbf{r}}]$. In a tele-operated system it is often useful to scale the input velocity from the operator, $\mathbf{v}_\mathrm{op}$ before feeding it to the robot controller. If the input from the user is of an other form such as a force or position it must first be transformed to an input velocity, by for example scaling or differentiating.

Thus the control signal $\mathbf{u}$ would be computed from the operator's input velocity $\mathbf{v}_\mathrm{op}$ as shown in (2.23). If $c = 1$ there exists a one-to-one mapping between the operator and the slave robot.

$$\mathbf{v} = c \cdot \mathbf{v}_\mathrm{op} \tag{2.23}$$

If the constant c is replaced by a diagonal matrix $\mathbf{C}$ it is possible to adjust the compliance independently for different dimensions of $\dot{\mathbf{x}}$. For example, setting the first three elements on the diagonal of $\mathbf{C}$ to $c \neq 0$ and all other elements to zero would result in a system that only permits translational motion and not rotation. This would be an example of a hard virtual fixture that constrains the motion from $\mathbf{x} \in \mathbb{R}^6$ to $\mathbf{p} \in \mathbb{R}^3$. If the rest of the elements on the diagonal were set to a small value, instead of zero, the fixture would be soft, allowing some motion in the rotational directions.

To express more general constraints assume a time-varying matrix $\mathbf{D}(t) \in \mathbb{R}^{6\times n},\ n \in [1..6]$ which represents the preferred direction at time t. Thus if $n = 1$ the preferred direction is along a curve in $\mathbb{R}^6$. Likewise, $n = 2$ would give preferred directions that span a surface. From $\mathbf{D}$ two projection operators can be defined (Marayong *et al.*, 2003), the span and kernel of the column space:

$$\mathrm{Span}(\mathbf{D}) \equiv [\mathbf{D}] = \mathbf{D}(\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T \tag{2.24}$$

$$\mathrm{Kernel}(\mathbf{D}) \equiv \langle\mathbf{D}\rangle = \mathbf{I} - [\mathbf{D}] \tag{2.25}$$

If $\mathbf{D}$ does not have full column rank the span cannot be computed, consequently it is better to compute (2.24) by using the pseudo-inverse (Marayong *et al.*, 2003), thus in practice the span is computed as:

$$\mathrm{Span}(\mathbf{D}) \equiv [\mathbf{D}] = \mathbf{D}(\mathbf{D}^T\mathbf{D})^{\dagger}\mathbf{D}^T \tag{2.26}$$

where $\mathbf{D}^\dagger$ denotes the pseudo-inverse of $\mathbf{D}$.

If the input velocity is split into two components as shown in (2.28) it is possible to rewrite (2.23) as:

$$\mathbf{v} = c \cdot \mathbf{v}_\mathrm{op} = c\left(\mathbf{v}_\mathrm{D} + \mathbf{v}_\tau\right) \tag{2.27}$$

$$\text{with } \mathbf{v}_\mathrm{D} \equiv [\mathbf{D}]\,\mathbf{v}_\mathrm{op} \text{ and } \mathbf{v}_\tau \equiv \mathbf{v}_\mathrm{op} - \mathbf{v}_\mathrm{D} = \langle\mathbf{D}\rangle\mathbf{v}_\mathrm{op} \tag{2.28}$$

Next introduce a new compliance that effects only the non-preferred component of the velocity input and write the final control law as:

$$\mathbf{v} = c\left(\mathbf{v}_{\mathrm{D}} + c_\tau \cdot \mathbf{v}_\tau\right) = c\left([\mathbf{D}] + c_\tau \langle\mathbf{D}\rangle\right)\mathbf{v}_{\mathrm{op}} \tag{2.29}$$

Computing the Preferred Direction

To constrain the motion to a given subspace it is enough to supply a continuous time-varying matrix $\mathbf{D}(t)$ that spans that subspace. For example, $\mathbf{D} = [1\ 1\ 0\ 0\ 0\ 0]^T$ would constrain the motion to a line in the x, y-plane with a 45° angle from the x-axis.

It is also possible to make a guiding fixture that moves towards a target pose $\mathbf{x}_{\mathrm{T}} \in \mathbb{R}^6$ by using a control law $\mathbf{w} = f(\mathbf{x}, \mathbf{x}_{\mathrm{T}})$ such that by setting the control input $\mathbf{u} = \mathbf{w}$ the system will eventually converge to the desired pose, that is

$$\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}_{\mathrm{T}}. \tag{2.30}$$

Choosing $\mathbf{D} = \mathbf{w}$ in (2.29) gives a control law that guides the operator to the given target pose $\mathbf{x}_{\mathrm{T}}$.

As described above, using $\mathbf{D} = [1\ 1\ 0\ 0\ 0\ 0]^T$ constrains the motion to a set of lines in the x, y-plane. However, only the *direction* of motion is constrained thus motion is possible along all lines that are parallel to the desired direction. If there is a requirement to follow a certain reference trajectory there is need for additional constraints. To accomplish this, the preferred direction at time t must be modified to guide the end-effector towards the desired trajectory. This is similar to guiding towards a desired pose as described above. However, it is now necessary to combine the two types of guiding virtual fixtures described above. Assume the desired direction is described in $\mathbf{D}$ and that there exists a motion objective $\mathbf{x}_{\mathrm{T}}$ that describes the desired trajectory. Then define the distance to the motion objective $\mathbf{d} = f(\mathbf{x}, \mathbf{x}_{\mathrm{T}})$ as the vector pointing from the current configuration $\mathbf{x}$ to the current set-point on the motion objective. Then define a new preferred direction $\mathbf{D}_c$ as:

$$\mathbf{D}_c(\mathbf{x}) = \left(\frac{(1-k_d)\,[\mathbf{D}]\,\mathbf{v}_{\mathrm{op}}}{||\mathbf{v}_{\mathrm{op}}||} + k_d\langle\mathbf{D}\rangle\mathbf{d}\right),\ 0 < k_d < 1 \tag{2.31}$$

The constant k_d determines how fast the end-effector is moved towards the set-point on the motion objective. One problem with (2.31) is the division with $\parallel \mathbf{v}_{\mathrm{op}} \parallel$ which is undefined when no user input is available. In practice this can be solved by simply setting $\mathbf{D}_c = \emptyset$ when the user input is below a threshold. An other approach used in (Marayong *et al.*, 2003) is to use a scaled version of (2.31) as shown in (2.32). This is valid since projection is invariant to scale.

$$\mathbf{D}_c(\mathbf{x}) = \left((1-k_d)\,[\mathbf{D}]\,\mathbf{v}_{\mathrm{op}} + k_d||\mathbf{v}_{\mathrm{op}}||\langle\mathbf{D}\rangle\mathbf{d}\right),\ 0 < k_d < 1 \tag{2.32}$$

The components necessary to construct a control law for a virtual fixture can now be summarized:

- A motion objective $\mathbf{x}_\mathrm{T} \subseteq \mathbb{R}^6$
- A control law $\mathbf{w} = f(\mathbf{x}(\mathbf{t}), \mathbf{x}_\mathrm{T})$ that moves the end-effector to $\mathbf{x}_\mathrm{T}$ as $t \to \infty$ by setting the control input $\mathbf{u} = \mathbf{w}$.
- A way to compute the preferred directions $\mathbf{D}$ relative to $\mathbf{x}_\mathrm{T}$ in such a way that $\langle \mathbf{D} \rangle \mathbf{w} = 0$ if and only if $\mathbf{w} = 0$, i.e the end-effector satisfies the motion objective.

Given these components a virtual fixture that guides the end-effector towards $\mathbf{x}_\mathrm{T}$ and constrains the motion to a subspace of $\mathbf{x}_\mathrm{T}$ determined by $\mathbf{D}$ can be computed by using the control law in (2.29) with $\mathbf{D}$ computed from (2.32).

The control law in (2.29) considers only the guidance of the end-effector towards the motion objective under the constraint specified by $\mathbf{D}$. To implement a forbidden regions virtual fixture it will also be necessary to deal with boundary constraints.

The implementation of forbidden regions virtual fixtures can be greatly simplified using application specific knowledge such as the availability of analytical expressions for the constraints, whether the constraints can be approximated by simple geometric shapes and the required resolution. However, for complicated regions requiring high accuracy the computational complexity can become an issue.

Consider the case of sinus surgery where a medical instrument must be inserted through the nose into the sinus cavity, (Li and Taylor, 2004). In this case the forbidden regions can be extracted from pre-operative 3D images. The extracted forbidden regions for such task is of high complexity and requires high accuracy. A natural representation of such complex geometry is to use a large set $G = \{g_1, g_2, \ldots, g_N\}$ of small triangles to build a mesh that closely approximates the surface (Watt, 2000, pp. 27-44). In every time step of the control algorithm the point $\mathbf{p}_k$ on triangle g_k with shortest distance to the robot can be computed for all triangles in the set G. For each point $\mathbf{p}_k$ there exists a point $\mathbf{p}_{\mathrm{r},k}$ on the robot that is closest to $\mathbf{p}_k$. This yields a set of point pairs $P = \{(\mathbf{p}_1, \mathbf{p}_{\mathrm{r},1}), (\mathbf{p}_2, \mathbf{p}_{\mathrm{r},2}), \ldots, (\mathbf{p}_N, \mathbf{p}_{\mathrm{r},N})\}$. From the set of point pairs P a subset $P_\mathrm{c} \subseteq P$ can be extracted by selecting only those point pairs $(\mathbf{p}_k, \mathbf{p}_{\mathrm{r},k}) \in P$ for which $\| \mathbf{p}_k - \mathbf{p}_{\mathrm{r},k} \| < e_\mathrm{thresh}$ holds. That is, the set P_c contains all point pairs in P that are closer to each-other than a threshold e_thresh. The points in P_c are the *collision candidates*.

If high accuracy is required it is necessary to be able to perform the search for collision candidates efficiently. To avoid testing every possible combination the mesh representation of the objects involved (the forbidden regions and the robot) must be stored in a data structure that allows an algorithm to efficiently find the required collision candidates. Octrees or binary space partition trees (BSP-trees) are standard data structures commonly used in computer graphics and other domains to solve these kinds of problems, (Watt, 2000, pp. 51-56). Using an octree the triangle mesh can be divided into sections as seen in figure 2.11 where each cell contains a number of triangles. Using this technique the search for collision candidates can be greatly improved by discarding parts of the tree that cannot possibly contain any collision candidates. For example, if the point on the box

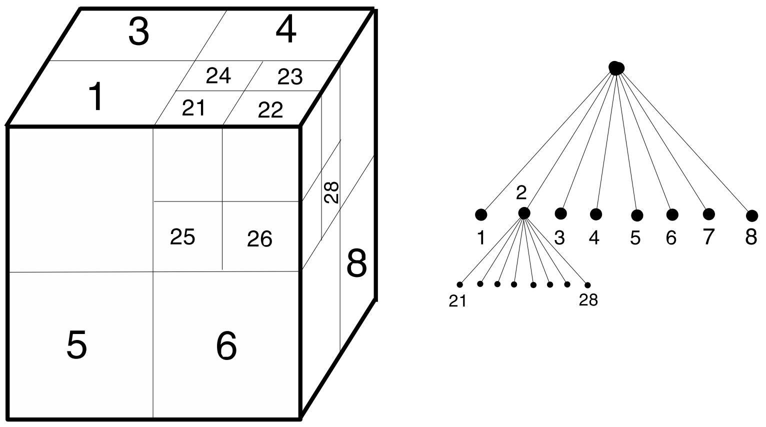


Figure 2.11. Example of an octree representation of a cubic space. On the right is the first two levels of the octree representation of the cubic space on the left.

labeled 1 in figure 2.11 that is closest to the robot is further away than e_{thresh} the whole box can be discarded at level 1 of the tree and that branch cut of from the search. The idea of BSP-trees is similar but they work by building a binary tree where each node represents a plane along one of the coordinate axis that partition the mesh representation into two parts. Actually in the original paper by Fuchs *et al.* (1980), the plane can have any orientation, but cutting along the coordinate axis simplifies the BSP-tree representation. In (Li and Taylor, 2004), a variant of octrees is used, a covariance tree, which is a k dimensional tree where each subspace is oriented in such way that the bounding boxes tend to align with the mesh surface, improving the efficiency of the search over a regular octree.

The collision candidates can then simply be used to constrain motion in directions that would decrease the distance between any point pair in P_c. For more details on this topic, see (Li and Taylor, 2004), that shows how to pose the combined guiding/forbidden regions virtual fixtures control law as an optimization problem.

Adjusting the Compliance

One issue with virtual fixtures that has recently received attention is how to set the compliance of the system, (Nolin *et al.*, 2003; Marayong *et al.*, 2002). Recall that the compliance determines how hard the fixture should be applied, thus a high compliance (low stiffness) supports isotropic motion while low compliance only allows motion along the preferred direction. The compliance can be adjusted by setting the variable c_τ in (2.29). Marayong *et al.* (2002) evaluates the effect of different compliance levels for path following, off-path targeting and obstacle avoidance. In path following the operator moves from A to B along the curve (see figure 2.12) with the aid of a virtual fixture. In off-path targeting the operator has to leave the path and move in a direction away from the fixture to touch point C

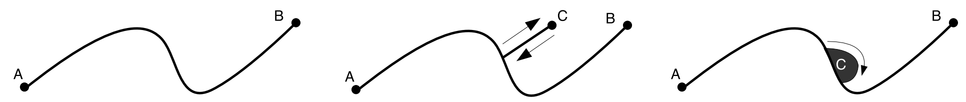


Figure 2.12. From left to right: curve following, off-path targeting and obstacle avoidance.

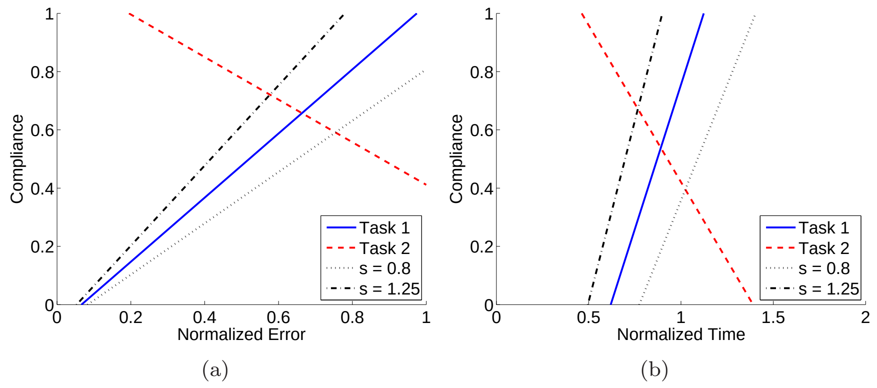


Figure 2.13. Optimal compliance selection between path following (task 1) and off-path targeting (task 2). (a) Compliance selected with respect to error (deviation from the path) and (b) compliance selected with respect to execution time.

and then move back to the path. When performing obstacle avoidance the operator first moves along the path until reaching the obstacle C. At this point the operator must leave the path to perform a motion around the obstacle and back to the path after clearing the obstacle. The virtual fixture applied was constructed to allow only motion along the reference curve. This means that in the off-path targeting and obstacle avoidance task the operator would work *against* the fixture. This also implies that with a hard fixture the off-path targeting and obstacle avoidance is impossible to perform.

Marayong *et al.* (2002) evaluates the error (deviation from the path) and execution time for different compliance levels and performs line-fitting to get the compliance as a function of the error and time respectively (Marayong *et al.*, 2002). Figure 2.13 shows a plot of the compliance as a function of execution time and error for the path following and off-path targeting. The optimal compliance is then chosen as the intersection between the two lines.

The next step is to introduce a task-oriented parameter s that can be used to assign a relative importance of freedom to guidance given a priori information about the task. The equation of the line representing task 1 in figure 2.13.a is $y = 1.1012x - 0.0731$, to assign more importance to guidance or freedom the task-oriented parameter is introduced to produce the following equation: $y = 1.1012x \cdot s - 0.0731,\ \ s \in [0, \infty[$. Setting $s > 1$ gives more importance to guidance while setting $s < 1$ gives more importance to freedom. The dash-dotted and dotted lines in figure 2.13 shows the difference obtained by introducing the task-oriented parameter.

In a similar way the relative importance of execution time vs tracking error can be accounted for. If the optimal compliance with respect to tracking error is denoted by c_e and the optimal compliance with respect to execution time is denoted by c_t the total compliance can be computed as $c = wc_\mathrm{e} + (1-w)c_\mathrm{t},\ w \in [0,1]$. The parameter w can then be used to set the relative importance of execution time vs tracking error. With $w = 0.5$ both execution time and accuracy is considered equally important. This method can be used to compute an optimal compliance with respect to the relative importance of different tasks as well as execution time and accuracy.

Even though an optimal compliance can be computed in terms of the relative importance of different tasks as well as execution time and accuracy using the above method, the main problem still remains. The fixture is only helpful during path following and it is actually counter-productive during off-path targeting and obstacle avoidance. To improve the situation Nolin *et al.* (2003) evaluates different methods for adjusting the compliance during task execution depending on whether the operator is trying to follow the nominal path or not.

In (Nolin *et al.*, 2003), three different methods for adjusting the compliance is evaluated, along with three different activation cues, that are used to determine when the fixture should be applied. Instead of the compliance Nolin *et al.* (2003) used the term *force scaling* to indicate the force applied by the fixture to drive the end effector towards the motion objective. This means that a low force scaling corresponds to high compliance and a high force scaling corresponds to a stiff fixture.

The tasks used for evaluating the compliance adjustment is similar to that used by Marayong *et al.* (2002) and consists of path following, off-path targeting and obstacle avoidance (see figure 2.12). To be able to adjust the compliance it is necessary to detect when the operator is no longer trying to follow the nominal path (the path along which the fixture is trying to guide the operator). In (Nolin *et al.*, 2003) three different (classes of) methods are evaluated; *explicit*, *implicit* and *automatic*. The methods works as follows:

Explicit: For the explicit activation cue the operator must manually inform the system that he/she is going to leave the nominal path. When the operator gives this signal, for example by pressing a foot-pedal, the fixture is activated. An advantage of the explicit method is that it is simple and robust. However it increases the mental load of the operator.

Implicit: The implicit activation cue works by detecting a certain predefined behavior of the operator, for example if no motion occurs for a predefined amount of time (so called silence). The implicit activation cue has the advantage of reducing the mental load of the operator compared to the explicit cue. However there is a risk of triggering the fixture at the wrong time if

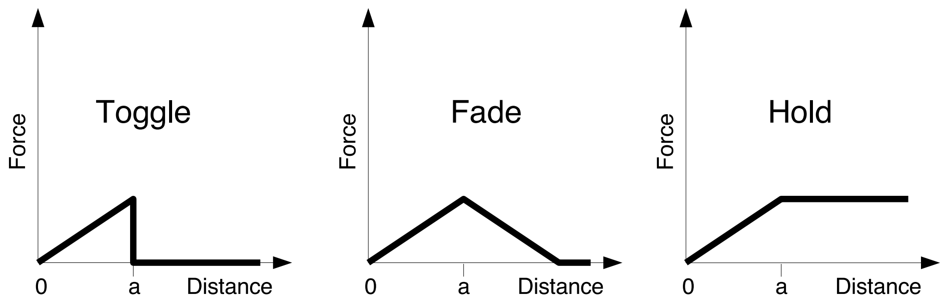


Figure 2.14. Force vs error for three different force scaling methods. An error of 0 means on the nominal path and a marks the activation point.

the implicit behavior is not carefully selected. Furthermore the use of silence increases the execution time of the task.

Automatic: When using automatic detection the activation cue a software monitors the operator's actions and tries to detect the intention of the operator. In (Nolin *et al.*, 2003) a simple method that activates the fixture at predefined locations is used. Advantages of automatic activation cues is that they reduce the mental load of the operator and does not increase the execution time. One problem with automatic activation cues is that they may be less robust and can be difficult to design.

Three methods for force scaling is also evaluated for a total of nine combinations. The force scaling methods used in (Nolin *et al.*, 2003) are: *toggle*, *fade* and *hold* and can be seen in figure 2.14. The force scaling methods works in the following way:

Toggle: The toggle method works so that when the operator moves away from the nominal path the force applied by the fixture increases linearly. When the activation cue is triggered the fixture is immediately turned of, effectively setting the stiffness to zero to allow isotropic motion.

Fade: When using the fade method for force scaling the compliance increases (force decreases) linearly after the point where the activation cue is triggered.

Hold: With the hold method the force produced by the fixture is held at a constant level once the activation cue has triggered.

After performing tests on a number of people Nolin *et al.* (2003) was not able to show with a statistical significance that any combination of activation cue and force scaling method was superior. Instead a number of interesting conclusions were presented: i) user preferences are key factors in selecting the force scaling method and activation cue, ii) the explicit and implicit activation cues do increase the mental load of the operator, iii) force scaling methods and activation cues must be combined appropriately and iv) the force scaling methods should be separated into moving a way from and returning to the virtual fixture. Nolin *et al.* (2003) does not deal with how the automatic activation cue can be implemented. This issue is dealt with in (Hundtofte *et al.*, 2002; Li and Okamura, 2003; Yu *et al.*, 2005) where HMMs are used to detect the intention of the operator. As mentioned before, if the operator's intention can be recognized it is possible, not only to use force scaling, but to apply a different fixture that actually aids the operator during the execution of the whole task. In chapter 3 a force scaling method is presented that sets the stiffness of the fixture proportional to the probability that the operator is following the fixture, this can of course only be done when automatic action cues are used.

2.3 Examples of Previous HMCS

Bettini *et al.* (2004) proposed a system for cooperative manipulation at millimeter to micrometer scale. Two different modes of operations were considered; *reference target*, where the desired path is a straight line segment in $\mathbb{R}^3$ from the current end effector position to a target point and *reference curve*, where the reference trajectory is a curve in $\mathbb{R}^3$. Although these two modes were devised with the application of retinal vein cannulation in mind there exists natural extension into other areas.

The virtual fixture control law used in (Bettini *et al.*, 2004) is the same as given by (2.29) with the exception that the input velocity is an input force. The span and kernel is computed from (2.24) where the inverse can be used because $\mathbf{D}$ is assumed to have full rank. For the reference curve case an additional error term is also introduced to return the end effector to the reference curve producing a control law similar to (2.32).

One issue raised in (Bettini *et al.*, 2004) is that when the end effector nears the target point $\mathbf{D}$ will not have full rank. This will unstabilize the system near the target point. This issue is solved by defining a sphere around the target point and altering the control law once this sphere is entered. One proposed alternative control law is to simply fix the preferred direction of the virtual fixture once the sphere is entered. Both hard virtual fixturing $c_\tau = 0$ and isotropic motion $c_\tau = 1$ is evaluated and it is shown that a hard fixture achieves the best performance. An alternative solution is also proposed, using an autonomous "snap-to" control law that simply brings the end effector autonomously towards the target point once the sphere is entered. However, this alternative is reported to be inferior to the cooperative control mode with a hard fixture.

The curve following task is evaluate with various degrees of compliance for the fixtures using the steady hand robot from Johns Hopkins University, (Taylor *et al.*, 1999; Abbott *et al.*, 2003). It is shown that a hard virtual fixture provides the best results w.r.t tracking error and execution time. However, this gives no room for the operator to handle modeling errors in the task or unforeseen events. It is also shown that even relatively soft fixtures dramatically improves the result compared to isotropic motion while at the same time allowing the operator to deviate from the reference curve if necessary.

Bettini *et al.* (2004) then extends the curve following to volumetric primitives where the reference curve mode is modified to allow motion inside a virtual tube spanning a safe volume, as seen in figure 2.15.

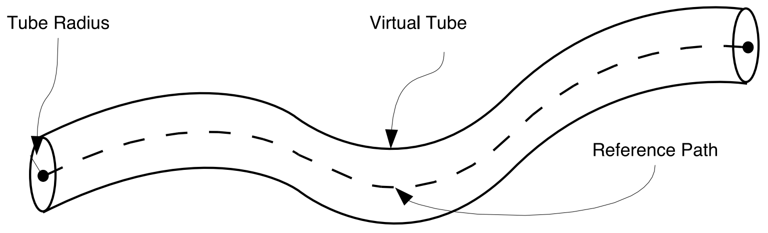


Figure 2.15. Virtual tube spanning a safe volume around a reference curve.

The reference target mode is also extended to volumetric primitives. In this case the corresponding primitive is an approach cone, as seen in figure 2.16. Similar to the one dimensional case, i.e. motion along a line in $\mathbb{R}^3$, the target point will be a singular point and a target sphere must be used to surround it in order to achieve a stable control.

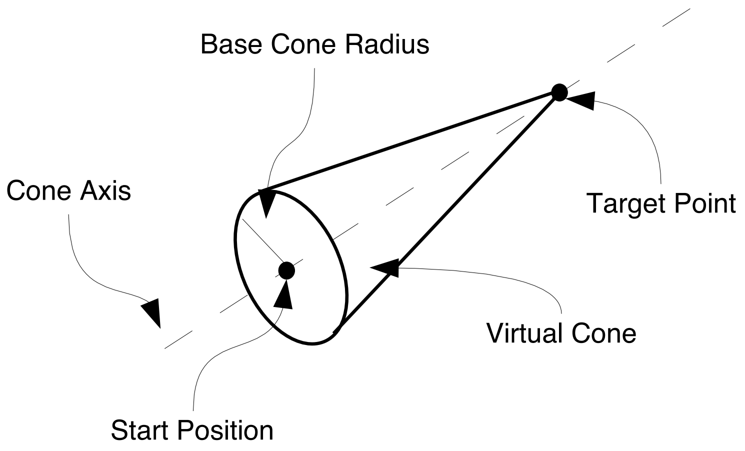


Figure 2.16. Virtual cone designed to guide the end effector towards a reference point.

The motion within the tube is unconstrained and when the end effector goes outside the tube the virtual fixture control law is applied to bring it safely back into the preferred region. However, this leads to a discontinuity at the surface which is handled by smoothing the transition from assisted to unassisted motion over an interval.

Finally (Bettini *et al.*, 2004) shows how to join tubes and cones to provide a safe approach to a target point as shown in figure 2.17. Each tube surrounds a locally differentiable reference curve and smooth transition between tubes, at possibly non-differentiable points, is achieved by smoothing.

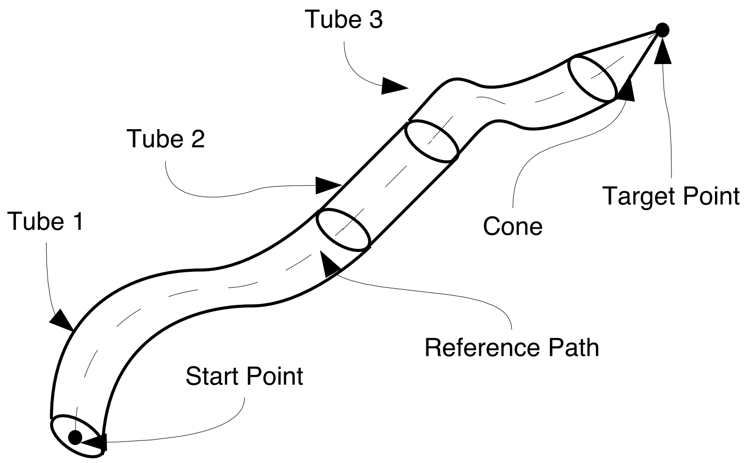


Figure 2.17. Combination of virtual tubes and cones designed to provide safe approach to a target point.

At the end (Bettini *et al.*, 2004) notes that one way to improve their system would be to incorporate models of the task and provide an estimate of the operator's intent in order to provide real-time and context-based assistance.

* * *

Hannaford and Lee (1990) used HMMs to model different tele-operation tasks. Two sequential tasks were considered, where at some point during the execution an event, random in one task and as an effect of the outcome of an operation in the other, triggers. When this event triggers the operator must follow one of two possible paths to finish the task, see figure 2.18. The states $\{S1, S2, \ldots, S9\}$ in figure 2.18 corresponds to the operator's mental model of the task. A HMM is created with one state for each mental state of the operator. During task execution force, torque, position and orientation is recorded for a total of 12 scalar measurements at each time step. The idea is then to compute the parameters of the HMM given sensor traces of users performing the different tasks and then to have the HMM classify new sequences of data into a sequence of states as function of time, where each state corresponds to a mental state of the operator.

To accomplish this it is assumed that the operator is able to describe the sequence of mental stages that will be traversed during task execution to build an

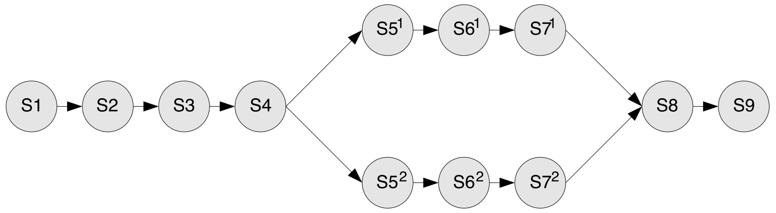


Figure 2.18. A sequential task that branches into one of two different paths after state 4.

appropriate HMM. By looking at sensor data and manually label the transitions between states the state transition probabilities of the HMM was computed according to (2.33). Where $\mathbf{A}_{i,j}$ is the probability of transition from state i to state j. When a branch is involved the probability has to be divided between branches according to the probability that a particular branch is selected.

$$\begin{aligned} \mathbf{A}_{i,i} &= \frac{\bar{t}}{\bar{t}+dt} \; \forall i \in [1..N] \\ \mathbf{A}_{i,i+1} &= 1 - \mathbf{A}_{i,i} \; \forall i \in [1..N] \\ \mathbf{A}_{i,j} &= 0, \; \forall i \neq j, \; i \neq j-1 \end{aligned} \tag{2.33}$$

Equation (2.33) also shows an important property of the structure of the HMM model. Transitions are only possible to the current state and to the next state in a sequential manner. Therefore no backtracking or switching between branches is possible. In (Hannaford and Lee, 1990) the probability of observing a given sensor value in a particular state is modeled as a Gaussian where all sensor dimensions are assumed to be independent. Figure 2.19 shows an example with two sensors and four states. The ellipses shows the points of equal probability densities. Other approaches to accomplish this is to use probability estimators, (Castellani *et al.*, 2004; Boite *et al.*, 1994), or a codebook obtained by pre-processing and quantizing the data, (Yu *et al.*, 2005).

The work presented by Hannaford and Lee (1990) has four major drawbacks. First the operator must be able to express the mental stages of the task in the form of a SLR HMM. Also the data obtained during the recording face must be manually examined and segmented into an appropriate state. Using a SLR model does not allow for backtracking or error handling when something goes wrong. Finally the Viterbi algorithm is used only to analyze data offline. These issues have since been addressed in a number of papers, (Hundtofte *et al.*, 2002; Li and Okamura, 2003; Castellani *et al.*, 2004; Yu *et al.*, 2005).

* * *

Hundtofte *et al.* (2002) used HMMs at the gestem level as opposed to the task level. This means that basic interaction primitives are modeled by a HMM and the

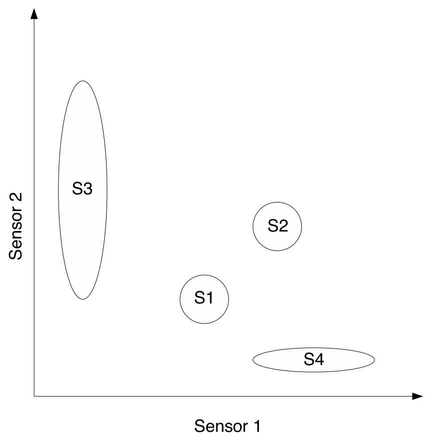


Figure 2.19. Example of probability densities under the assumption of Gaussian independent sensor values in a 2D sensor space. The figure shows four states and the corresponding probability densities of the two sensors.

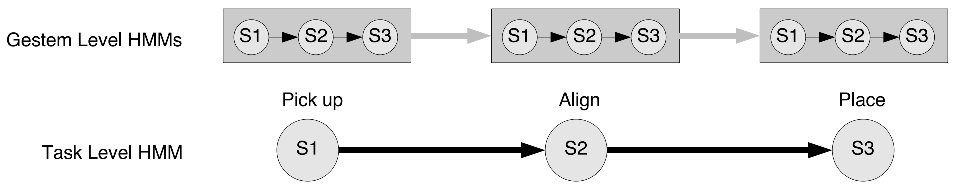


Figure 2.20. Example of a gestem level HMM (top) and a task level HMM (bottom). In a task level HMM each state corresponds to a sub-task. In the gestem level HMM there is one HMM for each sub-task and each state corresponds to some interpretation of the raw sensor data.

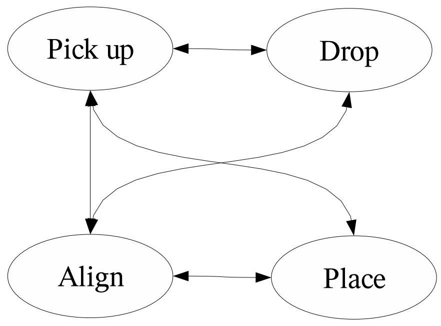


Figure 2.21. Several gestem level HMMs combined to form a network that captures a specific task.

task is represented as a network of such HMMs. Compare this to when the HMM is used to model the task, then each state in the HMM represents a particular sub-task, see figure 2.20. The idea presented in (Hundtofte *et al.*, 2002) is that it should be possible to derive a set of gestem HMMs that can be used across different tasks by combining them into an appropriate network. The system is evaluated on a peg-in-hole task and a painting (motion back and forth on parallel lines) task. The input to the system is 6 DOF force and torque an the size of the movement of the tool-tip during the last time step. Six different gestemes are used in (Hundtofte *et al.*, 2002). They are `place, position, insert, withdraw, remove` and `paint`. For each gestem a SLR HMM is trained, the number of states in each HMM was varied to find a suitable number for good representation.

Networks of these gestemes were then constructed to represent the different tasks, see figure 2.21. Note that there is no probabilities assigned to the transitions between nodes in the network, thus no *a priori* information on the sequence in which the sub-tasks are executed was provided. To train the system data was collected from test runs, where the operator manually segmented the data during execution by pressing a foot-pedal to signal task changes. The network of gestem HMMs are then used to segment the data offline. The segmentation of the HMMs has high probability to correspond well with that of the manual segmentation performed by the operator. No means for performing online recognition of intentions is presented in (Hundtofte *et al.*, 2002), nor is there any incorporation of knowledge of the sequence in which the sub-tasks are executed which could certainly be useful in more complex tasks.

* * *

Li and Okamura (2003) used HMMs for automatic segmentation and recognition of user motions. They present an algorithm for online recognition and use the result obtained from the online recognition to adjust the compliance of a virtual fixture. The system in (Li and Okamura, 2003) used force and position data recorded during a curve following task to detect and appropriately set the compliance of the virtual fixture depending on if the operator was trying to follow the curve or move away from the curve.

In the standard HMM approach the probability of observing a sequence of observation symbols $\mathbf{o} = \{o_1, o_2, \ldots, o_T\}$ given a HMM λ with N states is computed by (2.6). Equation (2.7) gives the probability of being in state j at time t, given the model λ. The total probability of the model is therefore simply the probability of being in *any* state at time T, as implied by (2.6). However, to use (2.6) it is necessary to know where to start counting, i.e. which observation symbol is the first one. A common approach is to use some form of change detector or manual segmentation to achieve this. For example, in speech recognition it is possible to use the silence between words as a cue for when to start the recognition. Similarly a foot pedal or a pause can be used to indicate a change of model in a tele-operation task. However, using a foot pedal or a pause to detected changes will interfere with the tele-operation task, thus it is desirable to overcome this limitation.

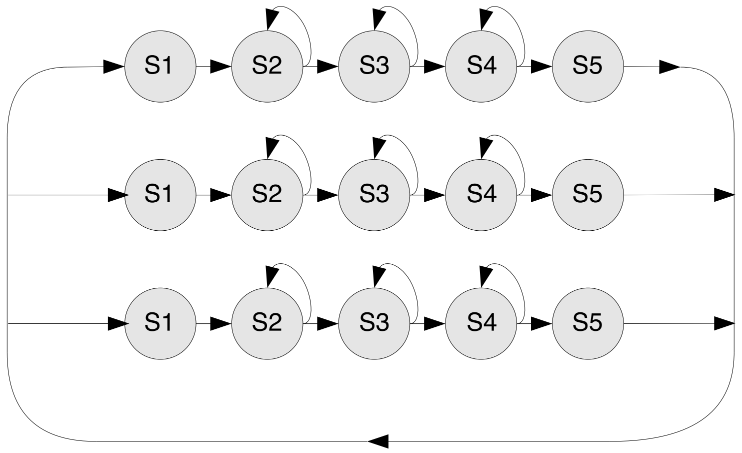


Figure 2.22. Online HMM recognition for three continuous HMMs. At any given time one of the HMMs is selected to describe the operator's motion.

The method proposed by Li and Okamura (2003) is to use a network of continuous online HMMs shown in figure 2.22. In the online approach presented by Li and Okamura (2003) all models are active in parallel. There are also two special states added to each HMM, the initial starter state and the resetting state (S1 and S5 in figure 2.22). When the system is initialized the alpha variables are initialized as shown by (2.34). That is, the probability of starting in state 1 is 1 for all models where as the probability of starting in any state other than 1 is 0.

$$\begin{aligned} &\alpha_1(0) = 1 \\ &\alpha_i(0) = 0,\ \forall\ 1 < i \leq N \end{aligned} \tag{2.34}$$

The probability of a given model is then computed by (2.6) with the exception that $\alpha_1(t)$ is computed according to (2.35), where $\alpha_{i,m}(t)$ is the probability of being in state i at time t given the mth HMM. This means that the probability that the model is in the initial state (i.e is reset) is the average of the probabilities that the M models are in their resetting state.

$$\alpha_1(t) = \frac{\sum\limits_{m=1,M}^{\alpha}{}_{N,m}(t)}{M} \tag{2.35}$$

Li and Okamura (2003) used three continuous online HMMs to analyze a path following task, where a HMM corresponded to one of the possible user actions: `silence, follow curve` and `avoid curve`. These HMMs were then used to detect, online, which of the actions the operator was performing and to switch on a virtual fixture if the follow curve actions had highest probability. The performance

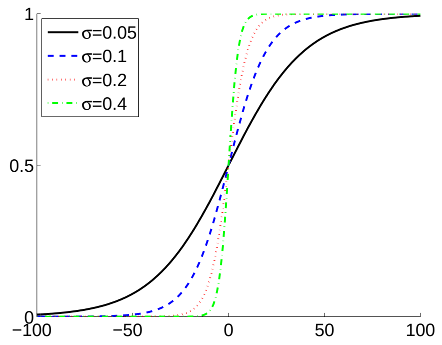


Figure 2.23. Sigmoid function used to transform the SVM distance measure into a conditional probability $P(\text{state } i|\mathbf{x})$. The choice of s determines the steepness of the function.

was then compared to the performance with a virtual fixture with fixed guidance and with no virtual fixture. It was shown that there was a significant improvement for the HMM approach over the others w.r.t either execution time or tracking error.

* * *

Castellani *et al.* (2004) used a hybrid HMM/SVM (**S**upport **V**ector **M**achine) approach to segment tele-operation data online and offline. Support vector machines were used as probability estimators for the HMM. SVMs are binary classifiers that separate two classes by an optimal separation hyperplane in a high dimensional space to which the input is transformed via a kernel function, see section 2.1.4.

The tele-operation task considered by Castellani *et al.* (2004) consists of a sequence of four different sub-tasks (where one sub-task can appear more than once). The different sub-tasks were: `move, tap, insert` and `extract`. The input to the system was one dimensional force data.

A HMM is used to describe the overall task, where one state in the HMM corresponds to a different sub-task. For each state in the HMM (sub-task) a corresponding SVM is computed in order to classify data as belonging to that state or not. Instead of using the SVM as a binary classifier Castellani *et al.* (2004) used the distance of a sample $\mathbf{x}$ to the margin of separation $d(\mathbf{x})$ (figure 2.4). This distance is then transformed to a conditional probability $P(\text{state } i|\mathbf{x}) \in]0,1[$ of membership of a state by transforming it through a sigmoid function, figure 2.23.

Instead of quantizing the input data and label it with an observation symbol the probability $P(\text{state } i|\mathbf{x})$ can be fed directly to the HMM without any prior assumption on the probability distribution of $\mathbf{x}$.

Castellani *et al.* (2004) used the HMM/SVM hybrid to segment a peg in hole task consisting of 11 sub-tasks in four categories. A HMM with 11 states was used to model the task and a SVM was trained for each of the four categories. The segmentation results reported by the hybrid method is somewhat higher than for similar tasks which uses a HMM with quantized observation symbols.

Chapter 3

Adaptive Virtual Fixtures

As mentioned earlier it has been demonstrated in a number of robotic areas how the use of *virtual fixtures* improves task performance both in terms of execution time and overall precision. However, the fixtures are typically inflexible, resulting in a degraded performance in cases of unexpected obstacles or incorrect fixture models. In this chapter, we propose the use of *adaptive virtual fixtures* that can cope with the above problems. A teleoperative or human-machine collaborative setting is assumed, with the core idea of dividing the task that the operator is executing into several sub-tasks. The operator may remain in each of these sub-tasks as long as necessary and switch freely between them. Hence, rather than executing a predefined plan, the operator has the ability to avoid unforeseen obstacles and deviate from the model. In this system each sub-task corresponds to following a trajectory. The probability that the user is following a certain trajectory (sub-task) is estimated and used to automatically adjusts the compliance. Thus, an online decision of how to fixture the movement is provided.

With recent changes in manufacturing such as just-in-time production, outsourcing and rapid process changes, it is necessary to be able to rapidly change the work-flow. Peshkin *et al.* (2001) and Moore Jr. *et al.* (2003) presents the concept of *Cobots* which are simple special purpose human-machine collaborative manipulation systems that have been used for automotive assembly in the car industry. Although frequently used, the Cobots have the following limitations: i) they are specially designed for one single purpose or task - thus one Cobot is required for every single task, ii) when the assembly task changes, the Cobots have to be modified to fit the new task, and iii) if new assembly tasks are introduced, new Cobots have to be constructed. Hence it would be interesting to use a standard robotic manipulator to assist humans in various tasks by providing virtual fixtures. However it may not be easy for an end-user such as a factory worker or even for medical staff to define the required span of the task.

In order to obtain a more flexible design framework, ideas are borrowed from the area of Programming by Demonstration (PbD). This enables building a system

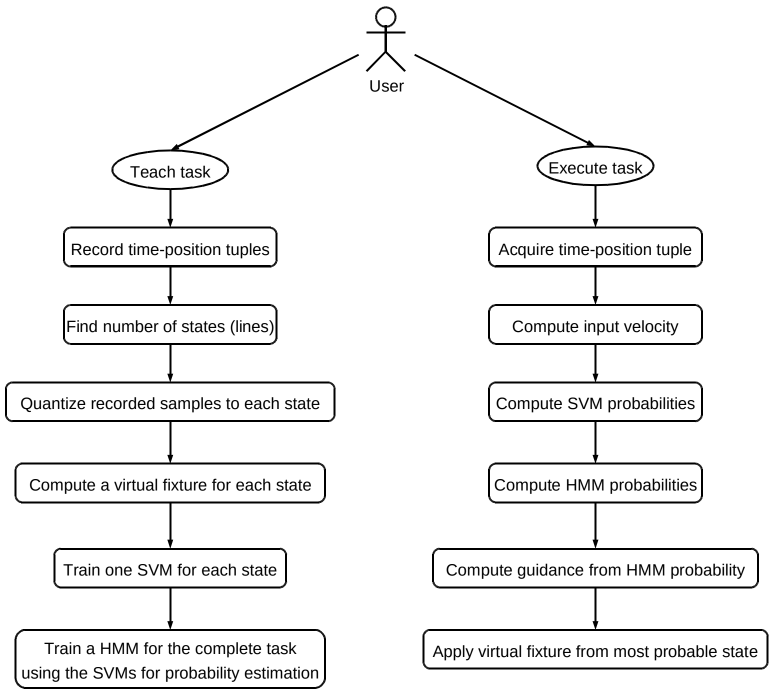

User
Teach task
Record time-position tuples
Find number of states (lines)
Quantize recorded samples to each state
Compute a virtual fixture for each state
Train one SVM for each state
Train a HMM for the complete task using the SVMs for probability estimation
Execute task
Acquire time-position tuple
Compute input velocity
Compute SVM probabilities
Compute HMM probabilities
Compute guidance from HMM probability
Apply virtual fixture from most probable state

Figure 3.1. Overview of the system used for task training and task execution.

where a human-machine collaborative manipulation system can be trained in a fast and easy way to increase the human's performance.

In this system, a high-level task is segmented to sub-tasks where each of the sub-tasks has a virtual fixture obtained from 3D training data. A state sequence analyzer learns what sub-tasks are more probable to follow each other. This is important for an online state estimator which estimates the probability of the user being in a particular state. A specific virtual fixture, corresponding to the most probable state is then automatically applied, see figure 3.1.

This chapter is organized in the following way. First the methods used to recognize sub-tasks are described. The recognition system is then evaluated on synthetically generated data to estimate the resulting performance of the system. After the evaluation it is described how the estimated state of the task can be used to provide fixturing of the motion. The complete system with sub-task recognition and fixturing is then evaluated on a trajectory tracking task with a real robot and a motion tracking device.

3.1 Recognizing Sub-Tasks

The first step towards achieving the adaptive virtual fixtures is to be able to recognize the current sub-task. In this case each sub-task corresponds to motion along a straight line in $\mathbb{R}^3$ and the task consists of a sequence of such motions. It is also necessary to be able to build a task model from training data, since the goal is to produce a system that can be used by an operator without explicit knowledge of how to define the necessary span of a task in order to generate a correct virtual fixture. In this case a SVM classifier will be used to estimate the probability of observations belonging to a certain state. The SVM classifier will then be used as a probability estimator for a task-level HMM that produces the final classification. The partitioning of the task into a suitable number of sub-tasks is performed by k-means clustering.

3.1.1 Retrieving Measurements

The input data consist of a set of 3D-coordinates that may be obtained from a number of modalities, describing a (position, time) tuple $\{\mathbf{q}, t\}$. From the input samples, movement directions are extracted by differentiating and normalizing. The noisy input samples are filtered using a dead-zone of radius δ around $\mathbf{q}$, i.e. a minimum distance δ since the last stored sample is required so that small variations in position are not captured. The input to the learning system is thus a sequence of normalized motion directions $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_N$ where $\parallel \mathbf{d}_i \parallel = 1,\ \forall i \in [1..N]$.

3.1.2 Estimating the Motion Directions

To estimate the nominal motion directions a number of example tasks are carried out. This results is a set of training sequences $\mathcal{O} = \{\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_N\},\ \mathbf{O}_i = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_{M_i}\}$. Each of these exemplars corresponds to a point on the unit sphere. K-means clustering (section 2.1.5) is now performed to find a number of distinct motion directions. The number of clusters to use can be manually selected or determined automatically by one of the approaches mentioned in section 2.1.5. The training data could of course also be manually segmented in which case the nominal motion direction would correspond to the mean value of the exemplars for a particular segment.

3.1.3 Estimating Observation Probabilities Using SVMs

For each nominal motion direction detected by the clustering algorithm, a SVM is trained to distinguish it from all the others (one-vs-all). In order to provide a probability estimation for the HMM, the distance to the margin from the sample to be evaluated is computed as (Castellani *et al.*, 2004):

$$f_j(\mathbf{x}) = \sum_i \alpha_i \cdot y_i \cdot \mathbf{x} \cdot \mathbf{x_i} + b \tag{3.1}$$

where $\mathbf{x}$ is the sample to be evaluated, $\mathbf{x}_i$ is the ith training sample, $y_i \in \{\pm 1\}$ is the class of $\mathbf{x}_i$ and j denotes the jth SVM. The distance measure $f_j(\mathbf{x})$ is then transformed to a conditional probability using a sigmoid function, $g(\mathbf{x})$, (Castellani *et al.*, 2004). The probability for a state i given a sample $\mathbf{x}$ can then be computed as:

$$P(\text{state } i|\mathbf{x}) = g_i(\mathbf{x}) \cdot \prod_{j \neq i} (1 - g_j(\mathbf{x})) \tag{3.2}$$
$$\text{where } g_i(\mathbf{x}) = 1/(1 + e^{-\sigma \cdot f_i(\mathbf{x})})$$

Given the above and applying Bayes' rule, the HMM observation probability $P(\mathbf{x}|\text{state } i)$ may be computed. The SVMs now serve as probability estimators for both the HMM training and state estimation. Since the standard SVMs do not cope well with outliers, a modified version of SVMs is used (Cortes and Vapnik, 1995).

3.1.4 State Sequence Analysis Using Hidden Markov Models

Even if a task is assumed to consist of a sequence of line motions, in an online execution step, the lines may have different lengths compared to the training data. When a certain line is followed, it is assumed that the corresponding line state is active. Thus, there are equally many states as there are nominal motion directions. Given that a certain state is active, some states are more likely to follow after depending on the task and, in our system, a fully connected Hidden Markov Model is used to model the task. The number of states is equal to the number of nominal motion directions found in the training data. The $\mathbf{A}$ matrix is initially set to have probability 0.7 to remain in the same state and a uniformly distributed probability to switch state. The π vector is set to uniformly distributed probabilities, meaning that all states are equally probable at the start time. For training, the Baum-Welch algorithm is used until stable values are achieved.

3.1.5 Evaluation

In this section the proposed learning and classification system will be evaluated. First the system will be tested on a relatively simple task, involving four distinct motion directions that are performed in a SLR fashion. The data for this example is generated synthetically in order to provide a ground truth. Then the system will be tested on a trajectory recorded using a motion tracking device.

The synthetic motion data is generated in the following way. First a reference task is generated by constructing a line sequence where each line has a length $l \in [\ell_{\min}, \ell_{\max}]$ and a randomly selected direction. Furthermore the angle θ between two consecutive lines is not permitted to be lower than a threshold $\theta_{\min}$. A reference trajectory T_r is then created by sampling the line sequence.

The simulated operator trajectories are then created in the following way. Given a reference trajectory T_r a target point $\mathbf{p}$ is selected on T_r so that the distance to $\mathbf{p}$

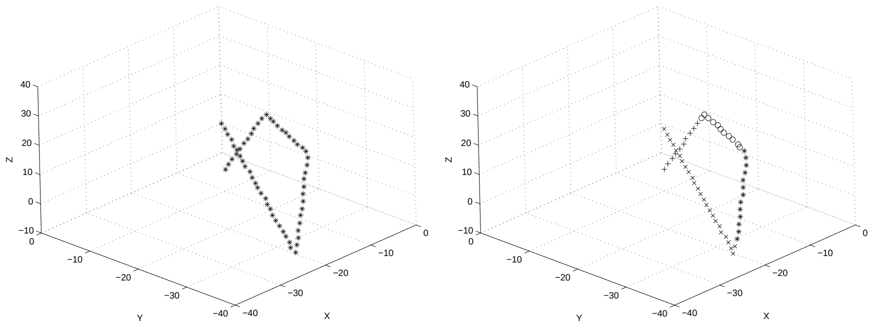


Figure 3.2. Left: Example of a training trajectory. **Right:** Classification of a trajectory after automatic segmentation and state sequence analysis. Different symbols indicate different states.

from the current position $\mathbf{q}$ is larger than some threshold ξ. A direction of motion $\mathbf{d}$ is then computed as the average between the direction towards $\mathbf{p}$ from $\mathbf{q}$ and the current direction of motion. A random error $\mathbf{e}_d$ is then added to $\mathbf{d}$ where each element of $\mathbf{e}_d$ is generated independently according to (3.3), where Γ is generated from a normal distribution ($\mu = 0, \sigma = 1$) and κ determines the noise level. Finally the current position $\mathbf{q}$ is updated by taking a step of size $\delta \cdot (1 + 2\kappa \cdot \Gamma)$ in the direction of $\mathbf{d}$ where δ determines the step size.

$$\mathbf{e}_d(i) = \kappa \cdot \Gamma \tag{3.3}$$

The simulated operator trajectories are finally normalized by sub-sampling or interpolation to form a training set Φ and a test set Ψ. A sample trajectory from the training set Φ is shown in figure 3.2 (left).

In this example six training trajectories are used. For each of the training trajectories the motion directions are extracted by differentiating and k-means clustering in conjunction with a CUSUM test that is used to find a suitable number of sates as described in section 2.1.5. For each state a SVM is trained to distinguish it from all the others and finally a HMM is trained to capture the sequencing of the task. The state transition probability for the HMM is initialized prior to training so that:

$$\begin{aligned} \mathbf{A}_{i,j} &= 0.7,\ \forall i = j,\ i,j \in [1..N] \\ \text{and } \mathbf{A}_{i,j} &= 0.3/(N-1),\ \forall i \neq j,\ i,j \in [1..N] \end{aligned} \tag{3.4}$$

where N is the number of states given from the k-means algorithm. After training the state probability matrix $\mathbf{A}$ has the values shown in figure 3.3 (left). In this example $N = 4$ states where detected.

The trajectories in the test set Ψ is now classified according the the trained model. An example of the segmentation of the trajectory into the four states is shown in figure 3.2 (right). The normalized likelihood of each state as a function of the sample number is shown in figure 3.3 (right). Figure 3.3 (right) also shows

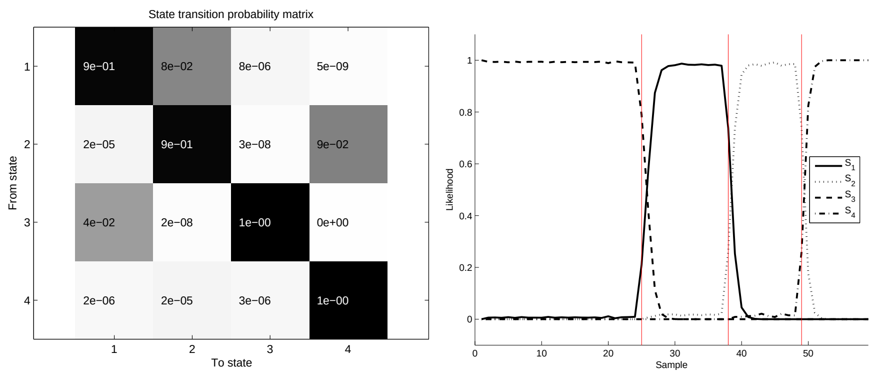


Figure 3.3. Left: **A** matrix of the example task. **Right:** Normalized likelihood of state as a function of the sample number. Vertical lines indicate the actual change times.

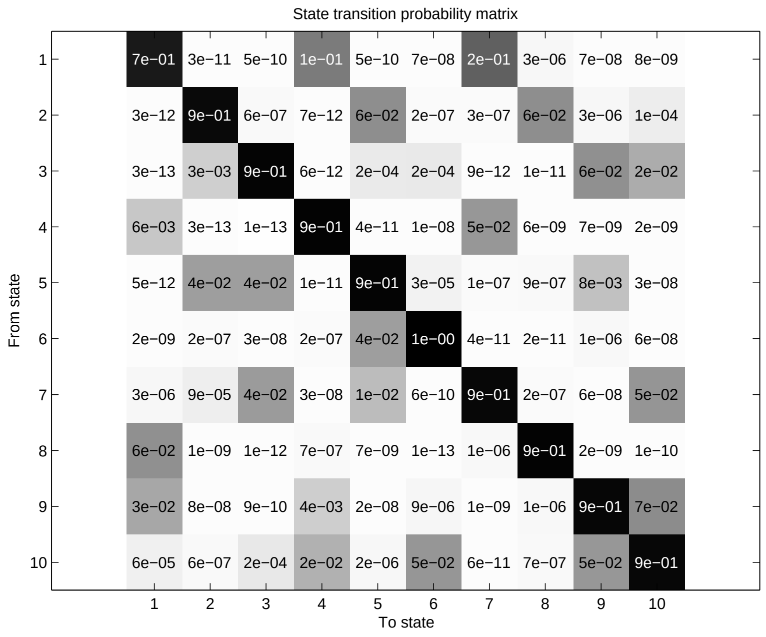
the actual change times from one state to the next as vertical lines. From this the delay for change can be estimated.

From this example a number of conclusions can be made. The initial state probability, π, was initially set to have uniform probability over all states. After training it was discovered that the whole probability mass had moved to state 3, giving it a probability of 1 as being the first state. Given this information it can be seen from the **A** matrix, shown in figure 3.3 (left), that a sequential model was extracted with the state sequence $3, 1, 2, 4$. This comes from the fact that these state transitions has a probability that is a factor $10^4 - 10^6$ larger than for any other transitions. However, since none of the state transitions has a probability of zero it is still possible to transition from any state to any other state given sufficient evidence in the form of observations, this makes the system robust to errors and allows it to handle task deviations.

Furthermore it can be seen from figure 3.3 (right) that the delay for detection is relatively small. The likelihood has gone from approximately zero to one in only 5 samples after the actual change time.

Data Generation

In the following example the motion data was generated with the help of a motion tracking device, called “Nest of birds” (NOB). The NOB is a magnetic tracker that consists of an emitter (figure 3.5(a)) and four pose measuring sensors (figure 3.5(b)). The emitter emits a magnetic field that is measured by the sensors. The pose, i.e. the position and orientation of each sensor can then be calculated. The sensors are mounted on a glove as shown in figure 3.5(c). The effective range of the emitter is approximately 1 m in all directions and data is sampled at roughly 30 Hz. The trajectory is taken as the position of sensor four, mounted on the index finger, during

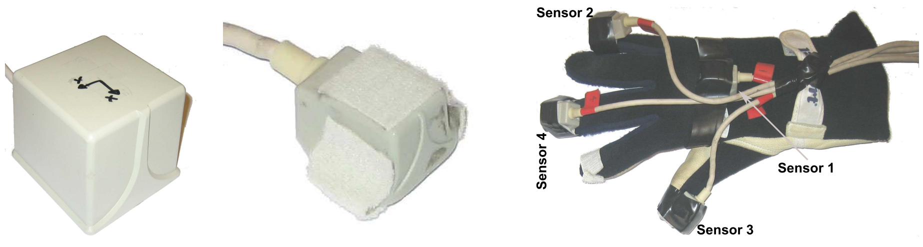


From state \ To state	1	2	3	4	5	6	7	8	9	10
1	7e−01	3e−11	5e−10	1e−01	5e−10	7e−08	2e−01	3e−06	7e−08	8e−09
2	3e−12	9e−01	6e−07	7e−12	6e−02	2e−07	3e−07	6e−02	3e−06	1e−04
3	3e−13	3e−03	9e−01	6e−12	2e−04	2e−04	9e−12	1e−11	6e−02	2e−02
4	6e−03	3e−13	1e−13	9e−01	4e−11	1e−08	5e−02	6e−09	7e−09	2e−09
5	5e−12	4e−02	4e−02	1e−11	9e−01	3e−05	1e−07	9e−07	8e−03	3e−08
6	2e−09	2e−07	3e−08	2e−07	4e−02	1e−00	4e−11	2e−11	1e−06	6e−08
7	3e−06	9e−05	4e−02	3e−08	1e−02	6e−10	9e−01	2e−07	6e−08	5e−02
8	6e−02	1e−09	1e−12	7e−07	7e−09	1e−13	1e−06	9e−01	2e−09	1e−10
9	3e−02	8e−08	9e−10	4e−03	2e−08	9e−06	1e−09	1e−06	9e−01	7e−02
10	6e−05	6e−07	2e−04	2e−02	2e−06	5e−02	6e−11	7e−07	5e−02	9e−01

Figure 3.4. **A** matrix of the example task.

a motion recorded with the NOB tracker. In this example four training sequences were recorded, they are shown in figure 3.6 (left). Once again the trajectories has been normalized before training. The motion directions and the result of the k-means clustering can be seen in figure 3.7 (left). In this example 10 states where detected. Once again the state transition probabilities were initialized according to (3.4) prior to training. After training the SVMs and the HMM the state transition probabilities takes the form shown in figure 3.4. The segmentation of a test trajectory can be seen in figure 3.6 (right) and the corresponding normalized likelihood as a function of the sample is shown in figure 3.7 (right).

A number of interesting conclusions can be drawn from this example. First of all it should be noted that the obtained segmentation into sub-tasks does not correspond to the "mental model" of the operator which would presumably be a sequence of unique states. However, this is not the purpose of this system. Here each state is defined as a motion along a unique direction and the purpose is to detect which state the operator is in at any given time to be able to provide the correct virtual fixture. The important thing to notice is that each time the operator expects a change of state, the system also changes state. The difference is that the operator tends to think of motion along parallel lines as two different states, whereas the system regards them as the same state, because the nominal motion direction is the same. However, since the same fixture should be applied this is not a problem.



(a) The emitter (b) A sensor (c) The data glove

Figure 3.5. The Nest of birds magnetic tracking system.

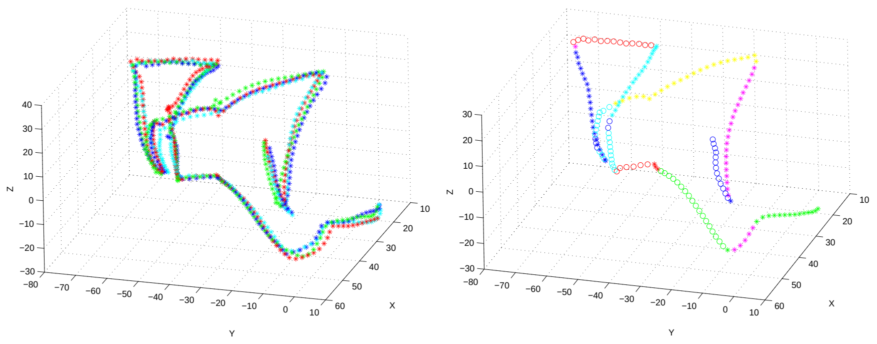


Figure 3.6. Left: Training trajectories. **Right:** Classification of a trajectory after automatic segmentation and state sequence analysis. Different symbols indicate different states.

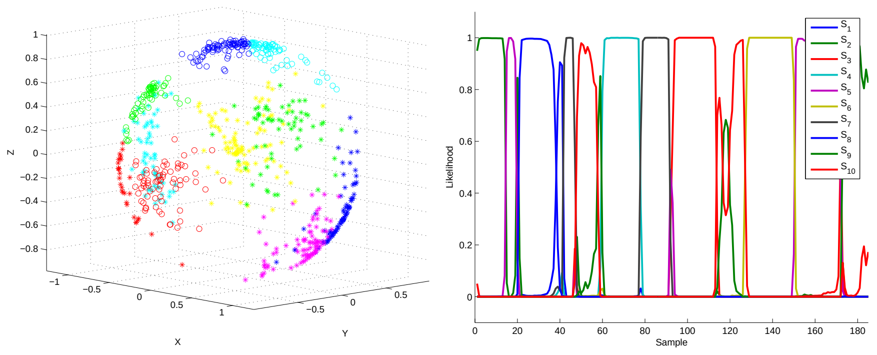


Figure 3.7. Left: Normalized likelihood of state as a function of the sample number. **Right:** Clusters as obtained from the automatic k-means algorithm.

Sometimes the system also divides a state, as perceived by the operator, into two or more states. Generating too many states is far better than generating too few, since this still means that the correct fixture can be applied all of the time. An example of this can be seen during the motion along the z-axis (light and dark blue circles) which is actually supposed, according to the operator, to be one and the same state, i.e. motion direction.

The reason for this behavior comes from the fact that the model is unconstrained and the HMM training simply chooses the parameters that locally optimize $\mathrm{P}(\mathbf{o}|\lambda)$ as described in section 2.1.2.

3.2 Fixturing of the Motion

Once it is possible to recognize the current sub-task it is of interest to provide the operator with guidance. Therefore a virtual fixture is associated with each sub-task. The virtual fixture is defined by the nominal motion direction of the associated sub-task, $\mathbf{d}$. In order to apply the correct fixture, the current state has to be estimated. The system continuously updates the state probability vector $\mathbf{p} = p_1, p_2, \ldots, p_N$, where $p_i = P(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t|\text{state } i)$ is calculated according to (3.5).

$$\begin{aligned}
\hat{p_i} &= \begin{cases} \pi_i \cdot P(\mathbf{x}|\text{state } i) & \text{if } \mathbf{p}^{\text{last}} = \mathbf{0} \\ P(\mathbf{x}|\text{state } i) \cdot \sum_j^N A_{ij} \cdot p_j^{\text{last}} & \text{otherwise} \end{cases} \\
p_i &= \hat{p_i} / \sum_k \hat{p_k} \qquad (3.5)
\end{aligned}$$

The state s with the highest probability p_s is chosen and the virtual fixture corresponding to this state is applied with the fixturing factor $k = \max(0.5,\ p_s \cdot \xi),\ \xi \in [0, 1]$, where $p_s = \max_i\{p_i\}$ and ξ is the maximum value for the fixturing factor. The fixturing factor describes how the virtual fixture will constrain the manipulator's motion by transforming the operator's commanded velocity, $\mathbf{v}_{\text{op}}$, to the robot's reference velocity $\mathbf{v}$ as shown in (3.6). In the case of a haptic input device, the fixture can also be used to provide the necessary feedback to the user and not only constraining the motion of the teleoperated device. Thus, when unsure which state the user is currently in, the user has full control. On the other hand, when all observations indicate a certain state, the fixturing factor k is set to ξ. This automatic adjustment of the fixturing factor allows the user to leave the fixture and move freely without having a special "not-following-fixture" state.

$$\mathbf{v} = \mathbf{proj_d}(\mathbf{v}_{\text{op}}) \cdot k + \mathbf{perp_d}(\mathbf{v}_{\text{op}}) \cdot (1-k) \qquad (3.6)$$

$$\text{where } \mathbf{proj_u}(\mathbf{a}) = \frac{\mathbf{a} \cdot \mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u} \qquad (3.7)$$

$$\text{and } \mathbf{perp_u}(\mathbf{a}) = \mathbf{a} - \mathbf{proj_u}(\mathbf{a}) \qquad (3.8)$$

3.3 Experimental Evaluation

In this section, three experiments are presented. The first experiment is a simple trajectory tracking task in a workspace with obstacles, shown in figure 3.8. The second is similar to the first one, but the workspace was changed *after* training, in order to test the algorithm's automatic adjustment to similar workspaces. In the last experiment, an obstacle was placed along the path of the trajectory, forcing the operator to leave the fixture. This experiment tested the adjustment of the fixturing factor as well as the algorithm's ability to cope with unexpected obstacles.

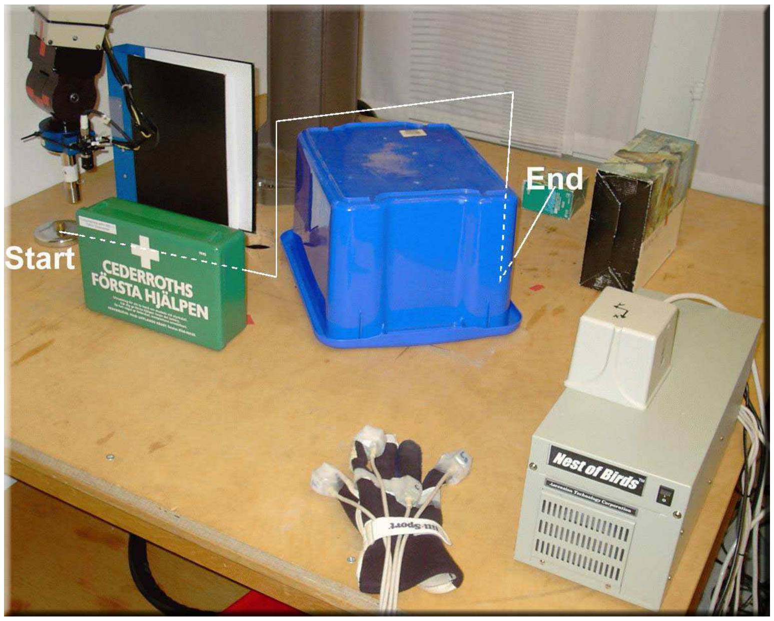


Figure 3.8. Typical workspace for pick-and-place task with obstacles. The white line shows the expected path of the end-effector.

In the experiments, a teleoperated setting was considered. A PUMA 560 robot was controlled via the NOB magnetic tracker which was mounted on a data-glove, as shown in figure 3.5(c), carried by the user. Once again only one sensor is used since it provides the full position and orientation estimate of the user's hand motion. Sub-Task recognition is performed with a frequency of 30 Hz. The movements of the operator measured by the NOB sensor were used to extract a desired input velocity to the robot. After applying the virtual fixture according to (3.6), the desired velocity of the end effector is sent to the robot control system.

The system also works well with other input modalities. For instance, a force sensor mounted on the end effector has also been used to control the robot. In all experiments, a dead-zone of $\delta = 2$ cm was used. This value of δ corresponds to the approximate noise level of our input device. One of the major difficulties of the system is that the input device provides no haptic feedback. Therefore, the virtual fixture framework is used to filter out sensor noise and correct unintentional operator motions. This is done by scaling down the input velocity that is perpendicular to the desired direction of the virtual fixture as long as the commanded motions is along the general direction of the learned fixture as described by (3.6).

In all experiments, a maximum fixturing factor was $\xi = 0.8$. A radial basis function with $\sigma = 2$ was used as the kernel for the SVMs and the value of σ in the sigmoid transfer function (3.2), was empirically chosen to 0.5. A PUMA 560 robot arm was used in all experiments.

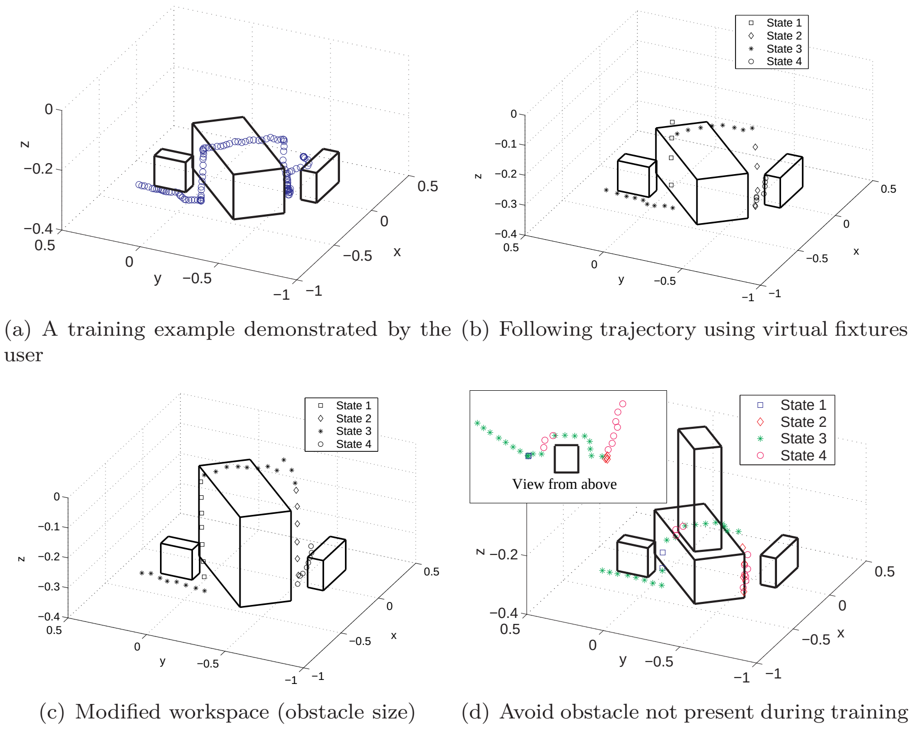
3.3.1 Experiment 1: Trajectory following

The first experiment was a simple trajectory following task in a narrow workspace. The user had to avoid obstacles and move along certain lines to avoid collision. During training, the operator demonstrated the task five times, the system learned from training data and four states were automatically identified. A typical training sequence can be seen in figure 3.9(a). The user then performed the task again, the states were automatically recognized and the robot was controlled aided by the virtual fixtures generated from the training data. The path taken by the robot is shown in figure 3.9(b). For clarity, the state probabilities and fixturing factor estimated by the SVM and HMM during task execution are presented in figure 3.10 (Left). This example clearly demonstrates the ability of the system to successfully segment and repeat the learned task, allowing a flexible state change.

Initially, the end-effector is moving along the y-axis, corresponding to the direction of state 3. Because of deviations from the state direction, the SVM probability will fluctuate since its estimation is based on the distance from the decision boundary. However the HMM probability remains steady due to the estimation history. This shows the advantage of using a HMM on top of SVM for state identification. At sample 24, the user switches direction and starts raising the end-effector. The fixturing factor decreases with the probability for state 3, simplifying the direction change. Then, the probability for state 1, corresponding to movement along the z-axis, increases. In total, the user performed 4 state transitions in the experiment.

3.3.2 Experiment 2: Changed Workspace

This experiment demonstrates the ability of the system to deal with a changed workspace. The same training trajectories as in the first experiment were used, but the workspace was changed after training. As it can be seen in figure 3.9(c), the size of the obstacle the user has to avoid has been changed. As the task is just a variation of the trained task, the system is still able to identify the operator's intention and correct unintentional operator motions. The trajectory generated from the online execution shows that the changed environment does not introduce any problem for the control algorithm since an appropriate fixturing factor is provided at each



(a) A training example demonstrated by the user (b) Following trajectory using virtual fixtures

(c) Modified workspace (obstacle size) (d) Avoid obstacle not present during training

Figure 3.9. End effector position in example workspace. The different symbols (colors) corresponds to the different states recognized by the HMM

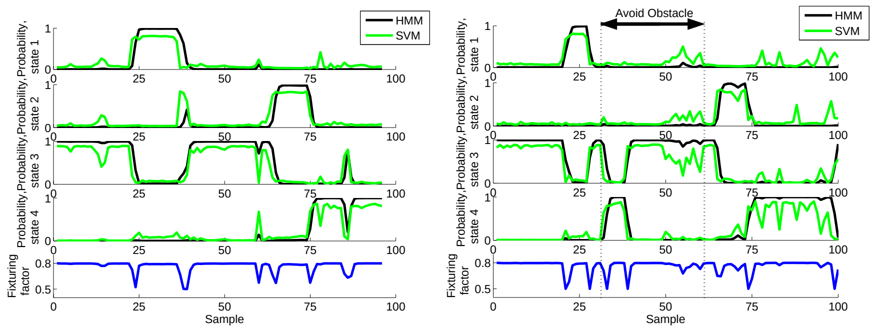


Figure 3.10. Estimated probabilities for the different states in experiment 1 (left) and experiment 3 (right). Estimates are shown for both the SVM and HMM. The fixturing factor is also shown

state. This clearly justifies the proposed approach compared to the work previously reported in (Peshkin *et al.*, 2001).

3.3.3 Experiment 3: Unexpected Obstacle

The final experiment was conducted in the same workspace as the first one. However, this time a new obstacle was placed right in the path of the learned trajectory, forcing the operator to leave the fixture. In this case, the virtual fixture is not aiding the operator, but may instead do the opposite as the operator wants to leave the fixture in order to avoid the obstacle. Hence, in this situation it is desired that the effect of the virtual fixture decreases as the operator avoids the obstacle. Once again, the same training examples as in the previous experiments were used.

figure 3.9(d) illustrates the path taken in order to avoid the obstacle. The system always identifies the class which corresponds best with input data. figure 3.10 (Right) shows the probabilities and fixturing factor for this experiment. Initially, the task is identical to experiment 1, the user follows the fixture until the obstacle has to be avoided. The fixturing factor decreases as the user diverts from the direction of state 3, and thus the user is able to avoid the obstacle. It can be seen that the overall task has changed and that new states were introduced in terms of sequencing. The proposed system not only provides the possibility to perform the task, but can also be used to design a new task model by demonstration if this particular task has to be performed several times.

3.4 Discussion

In this chapter, a system based on the use of *adaptive virtual fixtures* has been described. It is widely known that one of the important issues for teleoperative systems is the ability to divide the overall task into sub-tasks and provide the desired control in each of them. In particular, it has been shown that it is possible to use a HMM/SVM hybrid state sequence analyzer on multi-dimensional data to obtain an online state estimate that can be used to apply a virtual fixture. Furthermore, the process is automated to allow construction of fixtures and task segmentation from demonstrations, making the system flexible and adaptive by separating the task into sub-tasks. Hence, model errors and unexpected obstacles are easily dealt with.

In the current design, the algorithm automatically estimates the number of sub-tasks required to divide the training data. If instead it is possible for the user to manually select the number of states, the algorithm may be expected to perform even better. Such approach may be, for example, used in medical applications since surgical tasks are expected to be well-defined and known in advance. The experimental results have further shown that two straight lines with almost the same directions are classified as the same straight line. This, however, is not a severe problem as the fixture for this line is close to the optimal fixtures for the individual lines and appropriate guidance is provided by the system.

The proposed system have several limitations. For example, the proposed automatic detection of the states during training is only possible when the states corresponds to motion along straight lines. This is not such a severe problem as it may seem at first since many tasks can actually be decomposed into linear motions. In chapter 4 a method for performing motion intention recognition on general trajectories is presented. However, no automatic task segmentation during training is provided there and moreover, the proposed system is limited in that it only considers motion data. For some tasks it will be required to incorporate more sensing modalities such as force or vision to be able to distinguish between a broader set of sub-tasks.

Chapter 4

Layered HMM for Motion Intention Recognition

Hidden Markov models can be used on two levels when modeling human actions. A HMM can be used to recognize the operator's motion primitives, or *gestemes*, as in (Hundtofte *et al.*, 2002) or to model the mental stages of the operator performing a teleoperation task as in (Hannaford and Lee, 1990). A gestem-level HMM is used to recognize a primitive motion sequence and a task-level HMM is used to recognize a complete task. This chapter evaluates the HMM approach to gestem classification and proposes a layered HMM structure for motion intention recognition.

While the HMM/SVM approach presented in the previous chapter is applicable in a number of areas it may be too constrained for others. This is a price payed for allowing the method to be fully autonomous during training and execution. There are many settings where task knowledge and some understanding of task segmentation is available from the operator. In medical settings it may be possible to extract reference trajectories from pre-operative images obtained from, for example, a CT (Computed Tomography) scan. In other areas it is possible that the reference trajectory can be predefined and that it is only required to deviate from the predefined plan occasionally. In such settings a more complex task structure can be used to allow for more complicated sub-tasks. The idea here is to replace the SVM classifiers with the more expressive HMM classifiers in order to build a layered HMM where there is a HMM modeling the overall task and a HMM modeling each action primitive, hereafter referred to as gestemes.

The reason for using the LHMM instead of, for example, the HHMM (section 2.1.3) structure is that it corresponds well with the intended scenario. At the lowest level there are several models active in parallel classifying sensor data into action primitives. The classification then progresses through the LHMM until finally the task is modeled at the top level.

In this chapter no automatic segmentation of the task into sub-tasks is performed. Instead, the trajectories are manually labeled during a post-processing

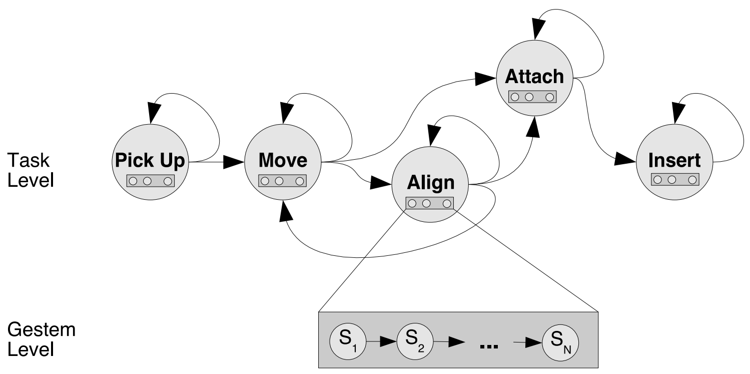


Figure 4.1. A two level layered hidden Markov model, modeling gestemes at level 2 and a task at level 1.

step. There are two main reasons for this. If the structure of the LHMM, e.g. number of sub-tasks etc, should be extracted from the training data, a huge amount of training sequences would be required. Furthermore, even if the structure of the LHMM could be extracted or predefined the Baum-Welch algorithm will adjust the HMM parameters λ to locally maximize the probability $P(Q|\lambda)$ of the state sequence Q, which may not correspond to the mental model of the operator. The training of the task level HMM is discussed further in section 4.1.2.

4.1 The Layered Hidden Markov Model

The concept of the Layered Hidden Markov Model (LHMM) is described in section 2.1.3. In this chapter a LHMM with two levels are considered. At level 1 a single HMM is used to model the task, where each state in the HMM corresponds to a sub-task. At level 2 there is a HMM for each of the K_2 possible gestemes that may occur during execution of the task. The observation sequence for the level 2 HMMs is generated from the quantized motion direction of the trajectory recorded during task operation. The index of the HMM with highest likelihood among the K_2 HMMs at level 2 is then taken to be the the observation symbol for the level 1 HMM. The level 1 HMM is then used to compute the probability of a certain state as a function of time given the observation sequence produced by the HMMs at level 2. Since each state in the level 1 HMM corresponds to a mental stage of the teleoperation task this information can be used to understand the operator's intention. The proposed structure is outlined in figure 4.1

Here, the winning HMM at level 2, i.e. the one with the highest likelihood, is chosen and an observation symbol corresponding to this gestem is generated for the level 1 HMM. The alternative would be to use the complete probability distribution and have the HMMs at level 2 act as a probability estimator for the level 1 HMM, as explained in section 2.1.3. However, according to (Oliver *et al.*, 2004) using the complete distribution does not give any apparent advantage over the simpler winner takes all model. Consequently, the simpler approach is taken in this work.

4.1.1 The Gestem HMM

The goal of the gestem HMMs is to distinguish between different motion primitives. For example there can be gestem HMMs to recognize motion along lines with different direction or circles with different radii and orientation in space. Actually the gestemes can be any arbitrary motion in 2D or 3D. The observations for the gestem HMMs are extracted from motion data. The trajectory is recorded, normalized and differentiated in order to compute the motion directions. The motion directions are then mapped to corresponding observation symbols in a way that will be described later in this section.

For the gestem HMMs three different approaches are evaluated. The one dimensional HMM, the multi dimensional HMM and the multi dimensional HMM with Fourier transform. The various HMMs and the generation of observation symbols are described below.

One dimensional HMM: The simplest HMM is the one dimensional HMM (OD HMM), where the observation symbols are then taken from a finite set $\mathcal{O} = \{O_1, O_2, ..., O_K\}$ of K discrete symbols. The $\mathbf{B}$ matrix is then used to retrieve the probability of observing the jth symbol in state i, that is $\mathbf{B_{i,j}} = P(O_j|\text{state}i)$, see section 2.1.2.

In this work the symbols are generated by k-means clustering of all the training directions. The number of cluster centers is 25 in all experiments, if not stated otherwise. This number was chosen by an offline examination of the data. The number of cluster centers is not crucial for the performance, but using too few clusters will make it hard to distinguish between different motion directions while using too many will make generalization difficult.

Multi dimensional HMM: The multi dimensional HMM, or MD HMM, assumes independence between the different dimensions of the input data. Thus there will be a $\mathbf{B}$ matrix for each dimension of the input data. This means that for a D dimensional HMM the observation symbols are also D dimensional where each dimension d contains values from a finite enumerated set, as described in section 2.1.2.

In this work, each dimension is split into 10 equally sized bins and the input directions are projected into these bins generating the observation symbols. As with the number of cluster centers the exact number of bins is not important but it has to be selected to facilitate discrimination and generalization.

Multi dimensional HMM with FT: The third type of HMM, the FFT HMM, considered in this paper is similar to the MD HMM except that instead of mapping the raw motion directions to symbols, each dimension of the raw input directions are pre-processed by applying the Fourier transform to small overlapping windows, similar to that reported in Yu *et al.* (2005). In this work a Hamming window, Harris (1978) of size 6 with 50% overlap was used.

4.1.2 The Task HMM

The task HMM, or the level 1 HMM in the LHMM structure, encodes the task sequencing information. Both levels of the LHMM work on the same time granularity and for each observation generated from the motion data the likelihood of the gestem (level 2) HMMs are computed. The gestem HMMs are enumerated and the index of the most likely gestem HMM is used as an observation for the task level (level 1) HMM.

Each of the states in the task level HMM corresponds to a sub-task in the operator's mental model and the most likely state can be computed in order to, for example, aid the operator with the execution of that sub-task.

It should be noted that there need not be a one-to-one mapping between a state in the task level HMM and a gestem HMM. Rather a specific gestem can correspond to different states depending on the previous state (the Markov assumption). Furthermore there may be several gestemes that can appear in a single state.

As mentioned previously the states of the task level HMM are supposed to correspond to the mental states of the operator. As a consequence, it is not possible to use the Baum-Welch algorithm to train the task level HMM, because it will optimize the HMM parameters λ in order to maximize $P(\mathbf{o}|\lambda)$ for the observation sequence $\mathbf{o}$. The approach taken in this work is to have the operator manually segment the trajectory into sub-tasks corresponding to the mental model of the operator. The gestem HMMs are then trained as before, using the Baum-Welch algorithm. Using the gestem HMMs to classify the training data a new observation sequence $\mathbf{o}'$ is obtained. From the observation sequence $\mathbf{o}'$ the $\mathbf{B}$ can be computed by counting the occurrences of each symbol in every state and the normalizing the rows of $\mathbf{B}$. The task level HMM can now be trained by a modified version of the Baum-Welch algorithm where the $\mathbf{B}$ matrix is kept constant.

4.2 Experimental Evaluation with Synthetic Data

To be able to better analyze and reproduce the results experiments are first performed on synthetic data. A reference task consists of a sequence of motion primitives randomly generated from two groups of motion primitives. The first group contains straight lines of varying directions and lengths and the second group is made up of circle segments with varying starting and ending angles as well as orientations and radii. Figure 4.2 shows typical simulated operator trajectories. These trajectory types may seem simple, but they were chosen because we believe that there exists several relevant tasks in areas such as medical surgery or automotive assembly that can be decomposed into a sequence of linear and circular motions.

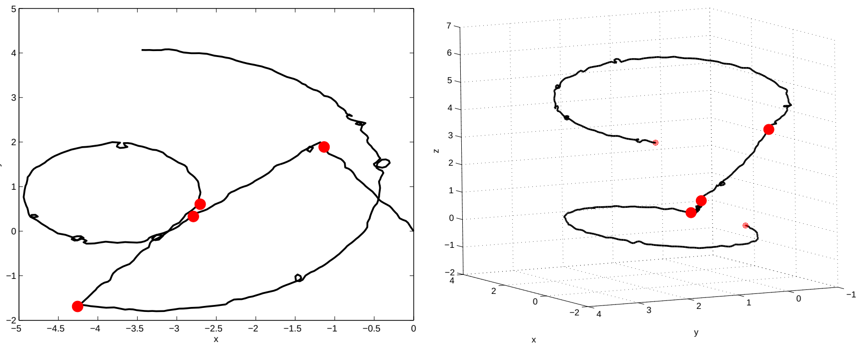


Figure 4.2. Typical simulated operator trajectories in 2D (left) and 3D (right). The red dots marks the change from one primitive to the next.

In section 4.3 similar experiments are performed with a robotic manipulator to verify the simulated results. There, the data is taken from a trajectory-tracking task where the end-effector of the manipulator is force controlled by a human operator.

The simulated operator trajectories are created in the same way as in section 3.1.5 where the step-size parameter δ was set to 0.05 in all experiments. The value of κ, that governs the amount of noise, was set to 0.15 in (3.3) for all experiments if not otherwise stated. In this chapter, three different classes of reference trajectories are used. They are referred to as *line*, *circle* and *mixed* trajectories in 2D respectively 3D. The line trajectories are made up of a sequence of linear segments, the circle trajectories are comprised of circle segments and the mixed trajectory type consists of a mixture of linear and circular segments.

In the experimental evaluation, three different types of HMMs are considered. The reason for evaluating different types of HMMs is that previous work has proposed the use of different kinds of HMMs and we are interested in investigating if there is an apparent advantage to any of them.

4.2.1 Experimental Evaluation

For the LHMM to be successful, there must be a robust underlying gestem classifier. Furthermore, the LHMM and gestem classifiers must be able to produce good results online with only partial observation sequences. The experimental evaluation in this part consists of evaluating the HMM gestem classifier for the three HMM types described in the beginning of this section with respect to the number of gestemes, the influence of the number of training samples, the effect of noise and the effect of the number of observation symbols.

The Gestem Classifier

The HMM is able to handle a large amount of noise as long as the noise is consistent during training and classification. To evaluate what amount of noise the gestem classifiers can handle, we tested the classification performance with several synthetic runs generated by varying the value of κ in (3.3) from 0.05 to 0.55. The input data was generated as described in Section 4.2, thus some gestemes can be very similar. If the gestemes are not generated at random but chosen from some set of gestemes that are constructed to be easy to distinguish between (such as the letters of the alphabet) the performance could be expected to be better than that reported here. For the proposed methods to work in the intended setting it is required to obtain good results with only a limited amount of training samples. Therefore only five training samples where used for the experiments in this section, if not otherwise stated. Furthermore, the results presented in this section are the average of 10 independent trials, if not explicitly stated.

Figure 4.3 and 4.4 shows the classification performance as a function of the noise variable κ in (3.3). From the figures, it is possible to conclude that a reasonable value for the noise parameter κ is less then $0.2 - 0.25$. For the remainder of the experimental results on synthetic data the value of κ is therefore set to 0.15 unless explicitly specified. It should be pointed out that setting the value of $\kappa > 0.7$ would prevent the simulated trajectories from reaching the goal without consuming unreasonable computer resources because the motion would be too dominated by noise. It is worth noting that already a value of $\kappa \in [0.3, 0.5]$ is almost as bad as guessing.

By examining the individual runs, it can be seen that the noise sensitivity is highly affected by the similarity of the gestemes. If the gestemes are similar, the performance decreases almost linearly with increased noise. If the gestemes contains few common symbols, the classification performance remains relatively unaffected until the noise starts to dominate (i.e is large compared to the nominal motion).

One interesting result of the noise experiment is that the OD HMM appears to have better performance with respect to noise sensitivity. We believe that the reason for this is the low dimensionality and that the k-means clustering of the pre-processing step helps with generalization since the cluster centers are affected by the actual training data instead of using pre-defined bins.

The second experiment evaluates the effect of the number of gestemes on classification performance. Remember that for every gestem there is a corresponding HMM which is trained to recognize it. As it can be seen in figure 4.5 and 4.6, the recognition performance drops almost linearly from 100% to about 60% for 25 gestemes for the medium noise case where $\kappa = 0.15$. Once again it is interesting to note that the OD HMM appears to have better performance w.r.t noise. The classification performance for the three dimensional data is a bit better but that can be explained with the fact that the individual gestemes are less likely to be similar.

It is well known that HMMs can be successfully trained with only a small

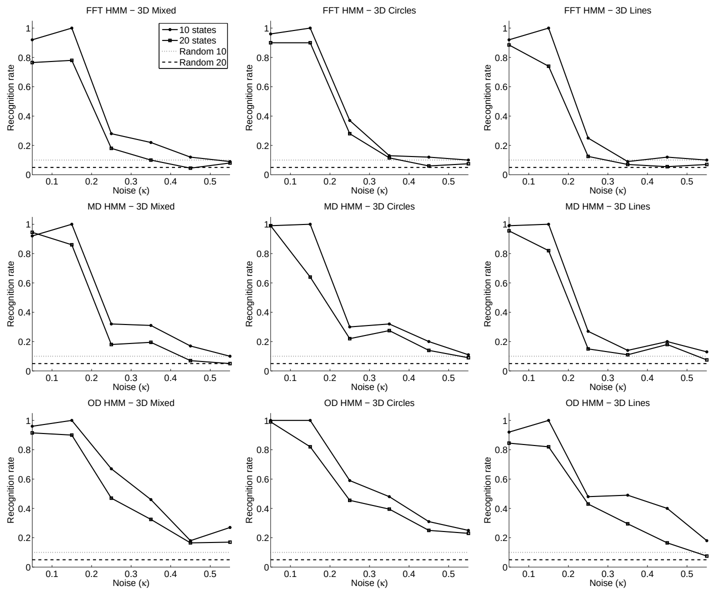


Figure 4.3. Classification performance as a function of noise.

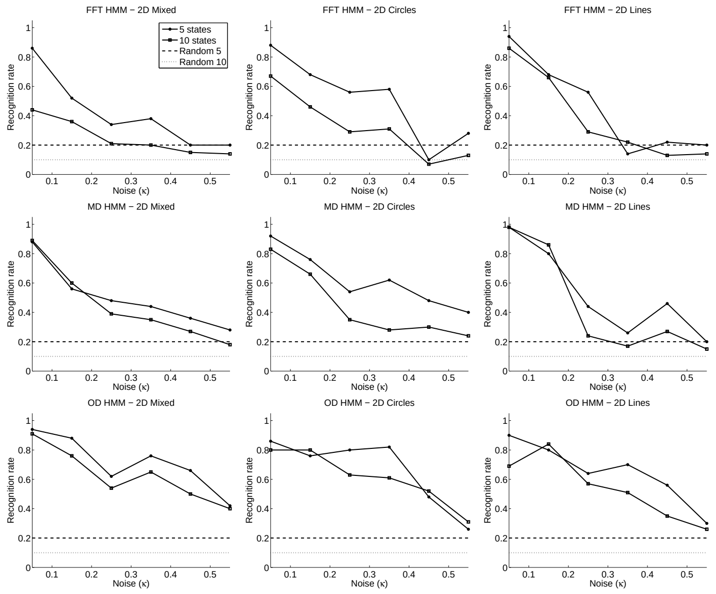


Figure 4.4. Classification performance as a function of noise.

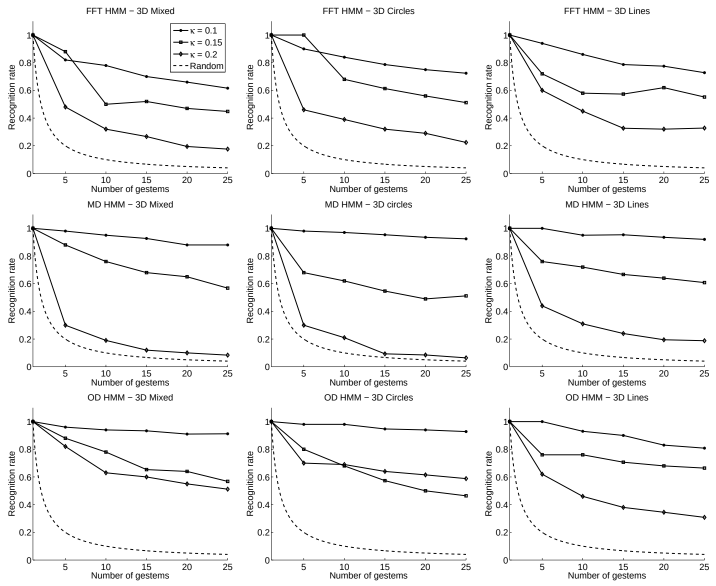


Figure 4.5. Classification performance as a function of the number of gestemes.

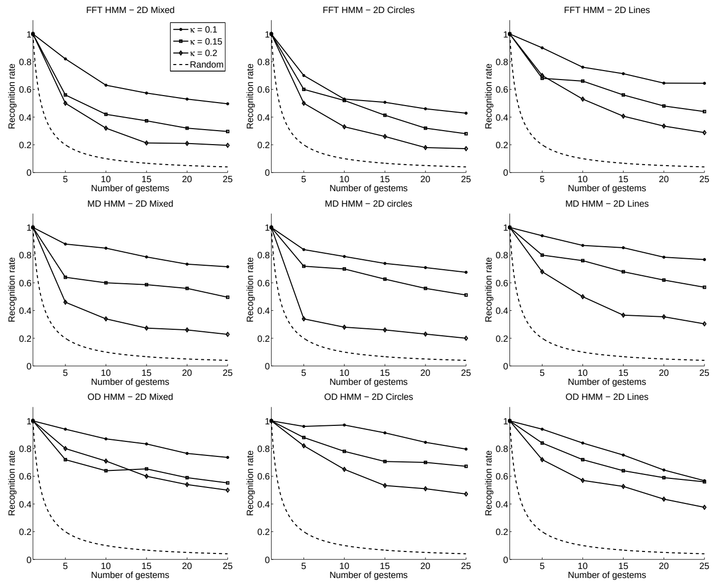


Figure 4.6. Classification performance as a function of the number of gestemes.

amount of training data. Especially if there are few outliers such as in our training data where the motion is perfect except for the introduced noise. It can be seen in figure 4.7 and 4.8 that the recognition rate is quite high even for only two training runs. This is a good feature of the HMM gestem classifier since in many settings extensive training is not possible. When the type of noise changes and outliers are introduced the necessary number of training sequences will increase in order to be able to capture the larger variations that occurs. However, preliminary results indicate that in practice the necessary number of training sequences is actually quite low as long as the training sequences are representative for what will occur during execution. Also it could be noted that with increased noise the number of required training sequences increases somewhat.

The number of distinct observation symbols is not crucial but have to be set reasonably. If too few symbols are used the HMM cannot distinguish between different directions leading to poor classification performance. At the same time, using too many symbols will prevent the HMM from generalizing, leading to poor classification performance because none of the models will correspond well with the training sequences. Figure 4.9 and 4.10 shows the classification performance as a function of the number of observation symbols. Remember that the observation symbols are defined differently between the OD and MD HMMs and values are thus not comparable. For the OD HMM the observation symbols correspond to the cluster centers obtain from the k-means clustering of the nominal motion directions of the training data, whereas for the MD HMMs the observation symbols are taken from $M \cdot D$ predefined bins of size $1/M$ giving a total of M^D different possible observations, where M is the number of discrete observation symbols and D is the dimensionality of the MD HMM.

So far, all the experiments have been conducted offline where the whole gestem was available. In order to work in the intended setting, the LHMM and gestem classifiers must be made to work online with only partially observed gestemes. However, because the performance is very dependent on the similarity between the first parts of the gestemes the success will vary depending on the type of task. Therefore we cannot present any quantitative data that is statistically interesting in the general case. Despite this, we can conclude from our experiments that given reasonably distinct gestemes the classification performance usually reaches its peak value approximately when 10%-20% of the gestem has been observed.

Another important aspect for online classification is the exact time at which the HMM recursion starts. In this case the exact times were known due to the fact that the test data was synthetically generated. If the change time for switching between gestemes are off there is a risk of observing very unlikely observation symbols and thus the correct HMM can be severely penalized in the beginning of the classification and in worst case never recover. There are ways around this problem, for example using the continuous HMM presented in Li and Okamura (2003). An alternative approach is based on a CUSUM test Gustafsson (2000) of the change in likelihood of the most probable model, i.e. the “elbow criterion”, as demonstrated in section 2.1.5.

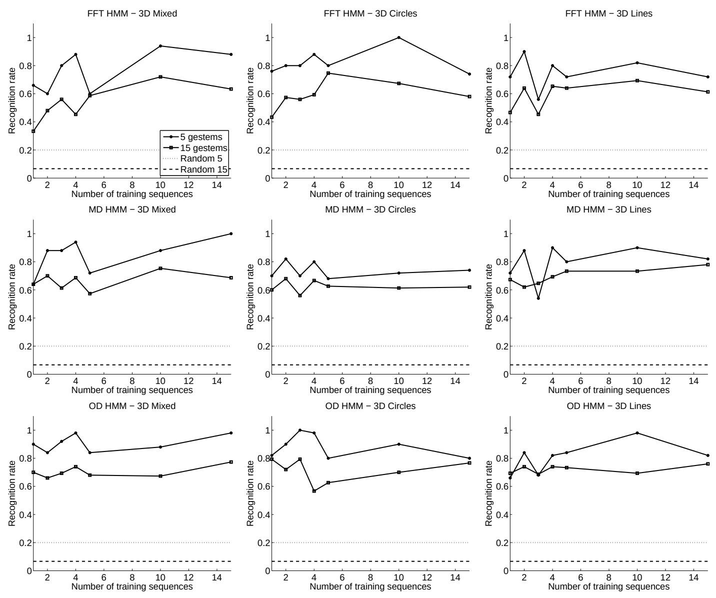


Figure 4.7. Classification performance as a function of the number of training sequences.

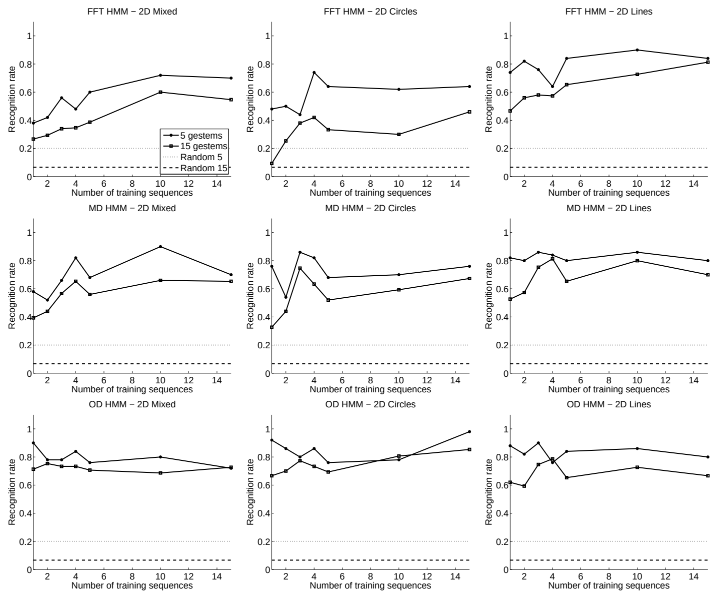


Figure 4.8. Classification performance as a function of the number of training sequences.

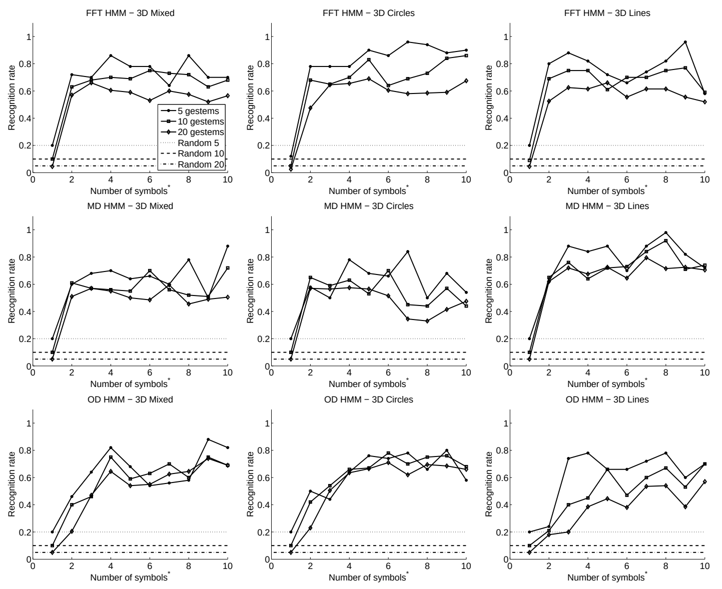


Figure 4.9. Classification performance as a function of the number of symbols. Note that the number of symbols have different meaning for one and multi dimensional HMMs.

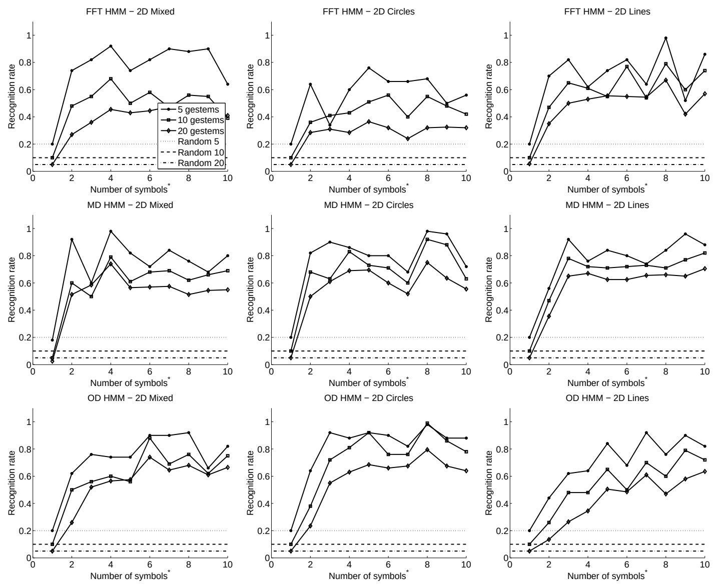


Figure 4.10. Classification performance as a function of the number of symbols. Note that the number of symbols have different meaning for one and multi dimensional HMMs.

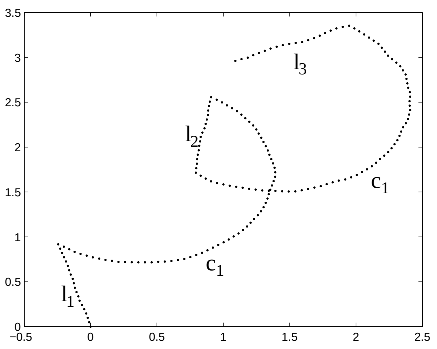


Figure 4.11. Example trajectory of a task with 5 states and 4 gestemes.

The LHMM

Figure 4.11 shows a 2D trajectory that contains four gestemes, $G = \{l_1, l_2, l_3, c_1\}$. The "mental model" of this task is that the gestemes should be performed in a sequential-left-to-right (SLR) fashion with the c_1 gestem appearing twice so the task should go through the five different states $S_1, ..., S_5$ and thus execute the gestemes in the following order: l_1, c_1, l_2, c_1, l_3. The gestem is exactly the same in S_2 and S_4 so one cannot differentiate between these states by simply monitoring the output from the four gestem classifiers.

A task level HMM is now trained on the output of the gestem classifiers. That is, the trajectory is classified by the gestem classifiers (online) and the sequence of winning gestemes are used as input to the task-level HMM which is trained in order to extract the task-model. figure 4.12 (bottom plot) shows a typical classification sequence obtained by the gestem classifiers. The vertical dashed lines indicate the switch from one state to the next. Note that there are only four gestemes recognized in the bottom plot whereas there are five states in the top and middle plots since the gestem c_1 is associated with two states.

It can be seen from figure 4.12 that even though the gestem classifiers are sometimes confusing gestem l_1 and c_1 the task-level HMM is still capable of determining the correct state. This is because the misclassifications of the gestem classifiers are consistent with training data and thus the task-level HMM expects some misclassifications. Furthermore the discriminant power of the LHMM is much better than that of the HMM, i.e. the difference between the most probable and the second most probable state is in general much larger for the LHMM.

4.3 Experimental Evaluation with a Robot System

In order to verify the validity of the proposed approach and to show that the quantitative results obtained with the synthetic data are relevant, we have performed

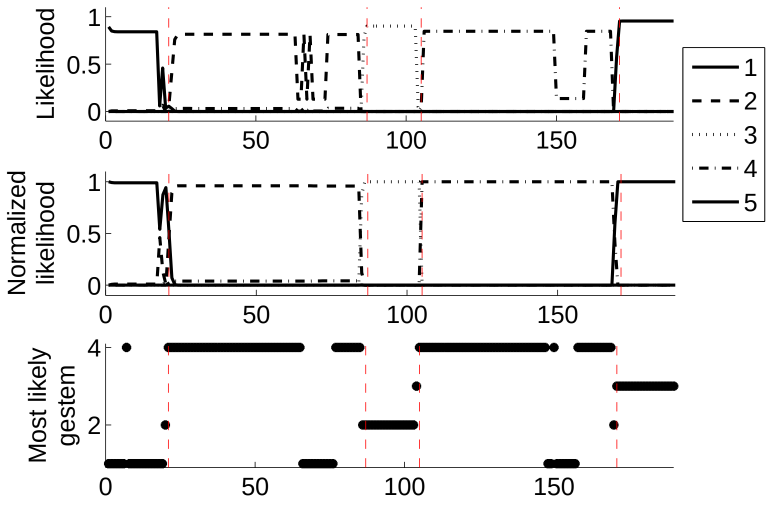


Figure 4.12. Classification of the LHMM for the task shown in figure 4.11. The top plot shows the likelihood of each state and the plot in the middle shows the normalized likelihood. The bottom plot shows the (online) classification of the motion by the gestem classifiers and is the input to the task-level HMM.

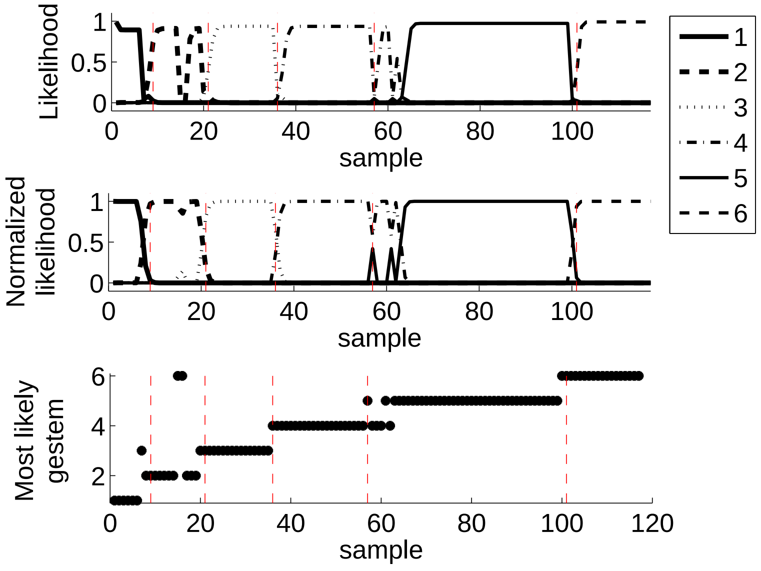


Figure 4.13. Classification of the LHMM for the 3D trajectory-tracking task executed with the robot manipulator as shown in figure 4.15 (right). The top plot shows the likelihood of each state and the plot in the middle shows the normalized likelihood. The bottom plot shows the (online) classification of the motion by the gestem classifiers and is the input to the task-level HMM.

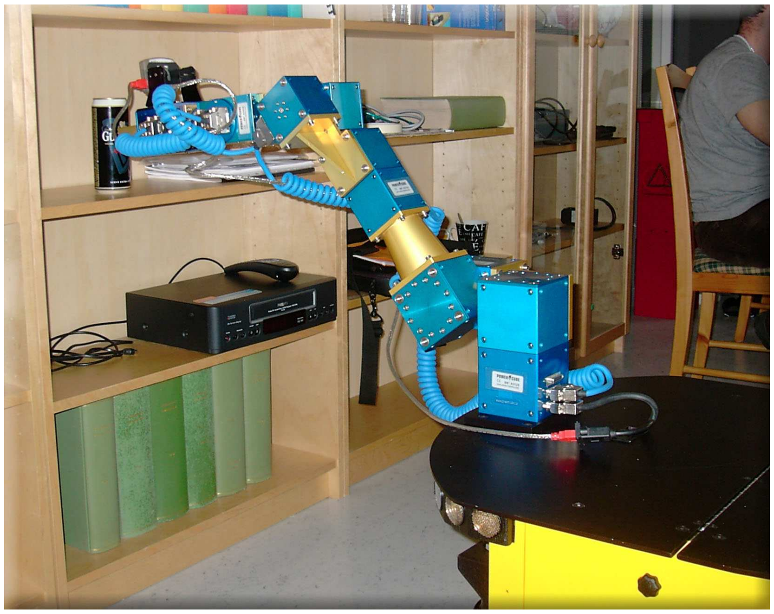


Figure 4.14. The manipulator used for the experimental validation.

a number of qualitative experiments with a robot manipulator. The robot used is an ActivMedia PowerBot and the manipulator used is made from a number of PowerCube elements and passive links and it is mounted on the mobile base as shown in figure 4.14.

The manipulator is equipped with a JR3 force/torque sensor mounted between the end-effector and the last link, providing 6 DOF force/torque measurements. It provides decoupled data at 8 kHz per channel, which is low-pass filtered with the bandwidth 30 Hz (-3 dB) by a DSP. The data is first read from the DSP and the current arm configuration is then used to subtract the influence of gravity on the end-effector. The force/torque vector is then transformed to the base frame attached to the base of the mobile platform. If the magnitudes of the force and torque are both below a threshold the velocity of all joints are set to zero. Otherwise, the Cartesian velocity of the arm is set to be proportional to the force. The same applies to the rotational velocities and the torque. The Cartesian velocities are then transformed to joint velocities of the arm using the inverse kinematics.

Due to the kinematics of the manipulator, large motions (of the joints) are sometimes required to realize small changes in orientation of the end-effector. This can make control of the manipulator more difficult than for a PUMA-like robot, that is, 6 rotary DOF with the 3 DOF of the wrist intersecting a single point. In all experiments the platform is stationary and the operator guides the manipulator by applying forces to the end-effector.

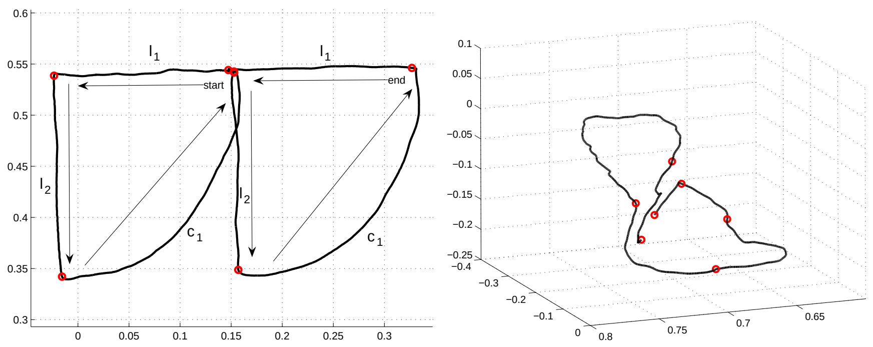


Figure 4.15. Representative trajectories for the two trajectory-tracking tasks. The dots mark the change between different states.

Two trajectory-tracking tasks are used in the experiments. The first task consists of tracking a sequence of lines and circle segments on a planar 2D surface, very similar to the simulated trajectories used previously. The second task consists of tracking a trajectory on an object in 3D without touching the object. Two representative trajectories are shown in figure 4.15.

The trajectories are normalized so that samples are 1 cm apart, this is a reduction of data of about 90%. From the normalized data the sequence of motion directions is computed and k-means clustering is used to identify 10, for 2D data, or 25, for 3D data, cluster centers used as the symbols in a one-dimensional HMM. The sequence of motion directions is then transformed to a sequence of observation symbols. A total of five trajectories were recorded and three of them were used for training and two for testing. The reason for using only five trajectories is that one important aspect of the proposed system is to provide good results even with a small amount of training data, which should be possible given the previously presented results.

Figure 4.13 shows the results of the online classification of the gestemes for the 3D trajectory in figure 4.15 (right).

As it can be seen the classification is good even though there were only three training trajectories available. One of the reasons for this is that it is the same person that performs the training and testing sequences. For operator independent training the number of required training samples is expected to be higher.

For the 2D case the LHMM was tested on the task shown in figure 4.15 (left) with the sequence of gestemes $\{l_1, l_2, c_1, l_2, c_1, l_1\}$. Even though the accuracy of the underlying gestem classifiers are very high the use of the LHMM is still motivated by two facts. First, it can encode the sequence of the gestemes and thus tell them apart even though the same gestem appears more than ones. Second the discriminate power is greater so it is possible to have a more confident classification. The same LHMM was also evaluated on the sequence $\{l_1, l_2, c_1, l_1, c_1, l_2\}$ which is *not* seen during training. It should be clear that the LHMM can still recognize the correct state sequence. However, there is a significant delay before the evidences (observations) are strong enough warrant a state change. This can be seen around sample 70 and 90 of figure 4.17.

It can be seen from figure 4.13 that even though the gestem classifiers are sometimes detecting the wrong gestem the LHMM can still clearly recognize the correct state. It can also be seen from the unnormalized likelihood that the total probability drops rapidly when unexpected gestemes are identified. This can be used to assign a measure of the certainty of the system and can be useful determining how much confidence to put into the classification during, for example, a fixturing of the motion, as was done in chapter 3. It should be noted that the LHMM was tested on manually segmented data and thus the classification is restarted at the "perfect" time which explains the zero delay for switching states and thus indicates the best possible results that can be achieved with the implemented system.

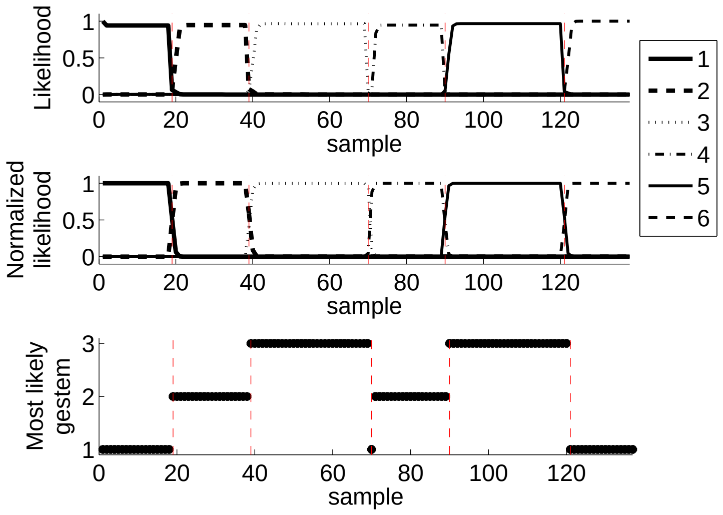


Figure 4.16. Classification of the trained sequence $\{l_1, l_2, c_1, l_2, c_1, l_1\}$. Classification of the LHMM for the two dimensional trajectory-tracking task. The top plot shows the likelihood of each state and the plot in the middle shows the normalized likelihood. The bottom plot shows the (online) classification of the motion by the gestem classifiers and is the input to the task-level HMM.

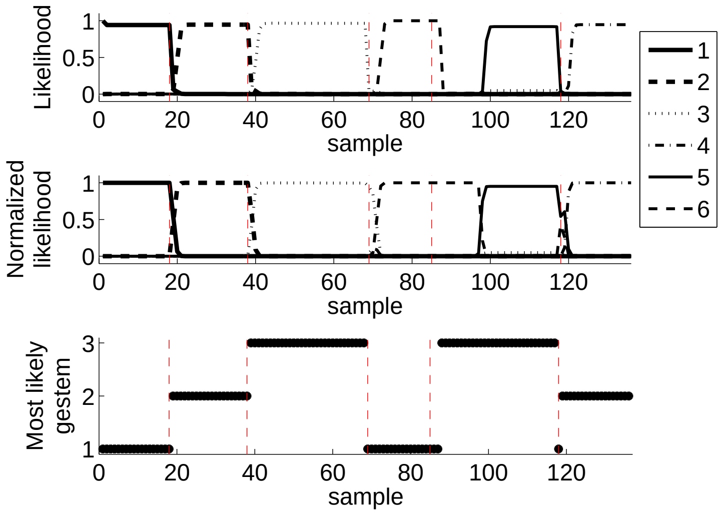


Figure 4.17. Classification of a sequence not seen during training, $\{l_1, l_2, c_1, l_1, c_1, l_2\}$. Classification of the LHMM for the two dimensional trajectory-tracking task. The top plot shows the likelihood of each state and the plot in the middle shows the normalized likelihood. The bottom plot shows the (online) classification of the motion by the gestem classifiers and is the input to the task-level HMM.

Chapter 5

Discussion and Future Work

This thesis focused on two aspects of human-machine collaborative systems. Classification of an operator's motion into a predefined state of a manipulation task and assistance based on virtual fixtures. The particular applications considered consisted of manipulation tasks where a human operator controls a robotic manipulator in a cooperative or teleoperated mode.

In **chapter 3** we proposed the use of *adaptive virtual fixtures* to cope with the problems of incorrect fixture models, handling of task deviations and automatic segmentation and learning of demonstrated tasks. A teleoperative or human-machine collaborative setting is assumed, with the core idea of dividing the task that the operator is executing into several sub-tasks. The operator may remain in each of these sub-tasks as long as necessary and switch freely between them. Hence, rather than executing a predefined plan, the operator has the ability to avoid unforeseen obstacles and deviate from the model. In our system, the probability that the user is following a certain trajectory (sub-task) is estimated and used to automatically adjusts the compliance. Thus, an online decision of how to fixture the movement is provided.

We have shown that it is possible to use a HMM/SVM hybrid state sequence analyzer on multi-dimensional data to obtain an online state estimate that can be used to apply a virtual fixture. Furthermore, the process is automated to allow construction of fixtures and task segmentation from demonstrations, making the system flexible and adaptive by separating the task into sub-tasks. Hence, model errors and unexpected obstacles are easily dealt with.

In the current design, the algorithm automatically estimates the number of sub-tasks required to divide the training data. If instead it is possible for the user to manually select the number of states, the algorithm may be expected to perform even better. Such approach may be, for example, used in medical applications since surgical tasks are expected to be well-defined and known in advance.

It should be noted that in the case with the automatic state segmentation the obtained sub-tasks does not correspond to the "mental model" of the operator which would presumably be a sequence of unique states. This is a consequence of the fully connected HMM used and is not a problem in the intended setting. Instead, each state is defined as a motion along a unique direction and the purpose is to detect which state the operator is in at any given time to be able to provide the correct virtual fixture. The important thing to notice is that each time the operator expects a change of state, the system also changes state. The difference is that the operator tends to think of motion along parallel lines as two different states, whereas the system regards them as the same state, because the nominal motion direction is the same. However, since the same fixture should be applied this is not a problem. Sometimes the system also divides a state, as perceived by the operator, into two or more states. Generating too many states is far better than generating too few, since this still means that the correct fixture can be applied all of the time.

In *chapter 4* we considered the use of a layered hidden Markov model (LHMM) to model human skills. We evaluated a gestem classifier that classifies motions into basic action-primitives, or *gestemes*. The gestem classifiers are then used in a LHMM to model a simulated teleoperated task. We investigated the classification performance with respect to noise, number of gestemes, type of HMM and the available number of training sequences. We also applied the LHMM to data recorded during the execution of a trajectory-tracking task in 2D and 3D with a robotic manipulator in order to give qualitative as well as quantitative results for the proposed approach.

Experimental evaluation shows that LHMMs have a good potential for modeling and real-time recognition of teleoperative and HMCS tasks. The evaluation has also shown that both one and multi dimensional HMMs are suitable for modeling gestemes and they are even able to handle gestemes that are quite similar in nature as long as the SNR is low. The HMMs are able to suppress relatively large amounts of noise as long as the noise is white. However, preliminary results indicate that the HMMs are more sensitive to other types of disturbances.

It is clear from the experimental evaluation that the LHMM has a strong potential to model complex tasks since it is able to perform well even with miss-classifications in the underlying layers. Thus as long as the gestem classifiers produce consistent misclassifications during training and testing the layered structure of the LHMM is able to handle this. The LHMM also has a much greater discriminating power than the standard HMM approach.

Furthermore, we have also pointed out three main issues that must be addressed in order to successfully build a layered HMM for online motion intention recognition. The first issue is that it is necessary to have a robust gestem classifier. Secondly, the gestem classifier must work online, with only partial observations of the gestem. Finally it is necessary to detect when one gestem ends in order to re-start the gestem classifiers.

This work presents a foundation for further development of a system where human intention recognition is used together with virtual fixtures and intelligent control to improve the execution of human-machine collaborative or teleoperative tasks. The key contributions of this work is the proposed LHMM structure for motion intention recognition and the associated evaluation together with the proposed method of using the probability that the operator is executing a certain state to adjust the compliance of a virtual fixture.

The short term goal of this research is to unify the ideas presented in chapter 3 and 4 to provide a framework with adaptive virtual fixtures applied to general operator trajectories. Furthermore, it is of interest to investigate the possibility to extend the intention recognition part to deal with more advanced features. As a first natural step orientation could be introduced, later tactile or force-torque sensors could be attached to the end-effector to provide information about contact forces during operation. The next step would be to incorporate computer vision into the system and extract task relevant features. It would also be interesting to further investigate the possibility to train the LHMM using unsupervised algorithms.

In the long run, it would be interesting to identify key areas where the application of fixturing combined with intention recognition is expected to have the highest impact. It would then be required to implement similar systems on the relevant hardware and perform user studies to analyze the possible improvements over the traditional approach. Obviously, safety concerns must be addressed before a system can be deployed outside of a laboratory setting.

Bibliography

J. J. Abbott, G. D. Hager, and A. M. Okamura. 2003. Steady-Hand Teleoperation with Virtual Fixtures. In *Proc. of the IEEE Int. Workshop on Robot and Human Interactive Communication*, pages 145 – 151.

M. Aizerman, E. Braverman, and L. Rozonoer. 1964. Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning. *Automation and Remote Control*, 25:821–837.

A. Bettini, P. Marayong, S. Lang, A. M. Okamura, and G. D. Hager. 2004. Vision-Assisted Control for Manipulation Using Virtual Fixtures. *IEEE Trans. on Robotics*, 20:953–966.

J-M. Boite, H. Bourlard, B. D'hoore, S. Accaino, and J. Vantieghem. 1994. Task Independent and Dependent Training: Performance Comparison of HMM and Hybrid HMM/MLP Approaches. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pages 617 – 620.

H. Bourlard and N. Morgans. 1990. A Continuous Speech Recognition System Embedding MLP into HMM. *Advances in Neural Information Processing Systems*, 2:186 – 193.

H. Bruyninckx and J. De Schutter. 1997. Where does the Task Frame go? In *Proc. of the International Symposium of Robotics Research*, pages 86 – 91.

C. J.C. Burges. 1998. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2:121–167.

A. Castellani, D. Botturi, M. Bicego, and P. Fiorini. 2004. Hybrid HMM/SVM Model for the Analysis and Segmentation of Teleoperation Tasks. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 2918–2923.

P-H. Chen, C-J. Lin, and B. Schölkopf. 2005. A Tutorial on ν-Support Vector Machines. *Applied Stochastic Models in Business and Industry*, 21:111–136.

C. Cortes and V. Vapnik. 1995. Support-Vector Networks. *Machine Learning*, 20: 273–297.

P. Dario, B. Hannaford, and A. Menciassi. 2003. Smart Surgical Tools and Augmenting Devices. *IEEE Trans. on Robotics and Automation*, 19:782–792.

A. Dielmann and S. Renals. 2004. Dynamic Bayesian Networks for Meeting Structuring. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pages V–629–632.

T. G. Dietterich. 2002. *Lecture Notes in Computer Science*, volume 2396, chapter Machine Learning for Sequential Data: A Review, pages 15–30. Springer.

R. Dugad and U. B. Desai. 1996. A Tutorial on Hidden Markov Models. Technical Report SPANN-96.1, Department of Electrical Engineering, Indian Institute of Technology, Bombay.

A. Elgammal, V. Shet, Y. Yacoob, and L. S. Davis. 2003. Learning Dynamics for Exemplar-Based Gesture Recognition. In *Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, pages 571–578.

S. Fine, Y. Singer, and N. Tishby. 1998. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32:41–62.

G. D. Forney Jr. 1973. The Viterbi Algorithm. *Proc. of the IEEE*, 61:268–278.

H. Fuchs, Z. M. Kedem, and B. F. Naylor. 1980. On Visible Surface Generation by A Priori Tree Structures. In *Proc. of the 7th annual conference on Computer graphics and interactive techniques*, pages 124 – 133.

A Ganapathiraju, J Hamaker, and J Picone. 2000. Hybrid SVM/HMM Architectures for Speech Recognition.

J-L. Gauvain and L. Chin-Hui. 1994. Maximum a Posteriori Estimation for Multivariate Gaussian Mixtureobservations of Markov Chains. *IEEE Trans. on Speech and Audio Processing*, 2:291–298.

R. M. Gray. 1984. Vector Quantization. *IEEE ASSP Magazine*, 1:4–29.

S. Günter and H. Bunke. 2003. Optimizing the Number of States, Training Iterations and Gausians in an HMM-based Handwritten Word Recognizer. In *Proc. of the 7th Int. Conf. on Document Analysis and Recognition*, pages 472– 476.

C. Guo, T-J. Tarn, N. Xi, and A. K. Bejczy. 1995. Fusion of Human and Machine Intelligence for Telerobotic Systems. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 3110 – 3115.

F. Gustafsson. 2000. *Adaptive Filtering and Change Detection.* John Wiley & Sons, Ltd.

B. Hannaford and P. Lee. 1990. Multi-Dimensional Hidden Markov Model of Telemanipulation Tasks With Varying Outcomes. In *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics*, pages 127–133.

F. J. Harris. 1978. On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform. *Proc. of the IEEE*, 66:51– 83.

C. S. Hundtofte, G. D. Hager, and A. M. Okamura. 2002. Building a Task Language for Segmentation and Recognition of User Input to Cooperative Manipulation Systems. In *Proc. of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 225–230.

K. Hyunsoo and P. Haesun. 2004. Prediction of Protein Relative Solvent Accessibility with Support Vector Machines and Long-Range Interaction 3D Local Descriptor. *Proteins: Structure, Function, and Bioinformatics*, 54:557–562.

T. Itoh, K. Kosuge, and T. Fukuda. 2000. Human-Machine Cooperative Telemanipulation with Motion and Force Scaling Using Task-Oriented Virtual Tool Dynamics. *IEEE Trans. on Robotics and Automation*, 16:505–516.

A. K. Jain, M. N. Murty, and P. J. Flynn. 1999. Data Clustering: A Review. *ACM Computing Surveys*, 31:264 – 323.

B-H. Juang and L. R. Rabiner. 1990. The Segmental K-Means Algorithm for Estimating Parameters of Hidden Markov Models. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 38:1639–1641.

M. Kaiser and R. Dillmann. 1996. Building Elementary Robot Skills from Human Demonstration. In *Proc. of the Int. Conf. on Robotics and Automation*, pages 2700–2705.

D. Kragić, P. Marayong, M. Li, A. M. Okamura, and G. D. Hager. 2005. Human-Machine Collaborative Systems for Microsurgical Applications. *Int. Journal of Robotics Research*, 24:731 – 741.

M. Li and A.M. Okamura. 2003. Recognition of Operator Motions for Real-Time Assistance Using Virtual Fixtures. In *Proc. of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 125– 131.

Ming Li and R.H. Taylor. 2004. Spatial Motion Constraints in Medical Robot Using Virtual Fixtures Generated by Anatomy. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1270– 1275.

R-H. Liang and M. Ouhyoung. 1998. A Real-Time Continuous Gesture Recognition System for Sign Language. In *Proc. of the Int. Conf. on Automatic Face and Gesture Recognition*, pages 558–567.

Henry C. Lin, Izhak Shafran, Todd E. Murphy, Allison M. Okamura, David D. Yuh, and Gregory D. Hager. 2005. Automatic Detection and Segmentation of Robot-Assisted Surgical Motions. In *Proc. of the Int. Conf. on Medical Image Computing and Computer-Assisted Intervention*, pages 802–810.

J. B. MacQueen. 1967. Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297.

P. Marayong, A. Bettini, and A. Okamura. 2002. Effect of Virtual Fixture Compliance on Human-Machine Cooperative Manipulation. In *Proc. of the IEEE/RSJ Int. Conf. on Inteligent Robots and Systems*, pages 1089– 1095.

P. Marayong, M. Li, A. M. Okamura, and G. D. Hager. 2003. Spatial Motion Constraints: Theory and Demonstrations for Robot Guidance Using Virtual Fixtures. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1270– 1275.

C. A. Moore Jr., M. A. Peshkin, and J. E. Colgate. 2003. Cobot Implementation of Virtual Paths and 3-D Virtual Surfaces. *IEEE Trans. on Robotics and Automation*, 19:347–351.

T. Mori, Y. Segawa, M. Shimosaka, and T. Sato. 2004. Hierarchical Recognition of Daily Human Actions Based on Continuous Hidden Markov Models. In *Proc. of the IEEE Int. Conf. on Automatic Face and Gesture Recognition*, pages 779–784.

Jason T. Nolin, Paul M. Stemniski, and Allison M. Okamura. 2003. Activation Cues and Force Scaling Methods for Virtual Fixtures. In *Proc. of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 404– 409.

N. Oliver, A. Garg, and E. Horvitz. 2004. Layered Representations for Learning and Inferring Office Activity from Multiple Sensory Channels. *Computer Vision and Image Understanding*, 96:163–180. ISSN 1077-3142.

S. Payandeh and Z. Stanisic. 2002. On Application of Virtual Fixtures as an Aid for Telemanipulation and Training. In *Proc. of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 18–23.

M. A. Peshkin, J. E. Colgate, W. Wannasuphoprasit, C.A. Moore, R. B. Gillespie, and P. Akella. 2001. Cobot Architecture. *IEEE Trans. on Robotics and Automation*, 17:377–390.

L.R. Rabiner. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. of the IEEE*, 77:257–286.

S. Renals, N. Morgan, H. Bourlard, M. Cohen, and H. Franco. 1994. Connectionist Probability Estimators in HMM Speech Recognition. *IEEE Trans. on Speech and Audio Processing*, 2:161–174.

C. N. Riviere, W. T. Ang, and P. K. Khosls. 2003. Toward Active Tremor Canceling in Handheld Microsurgical Instruments. *IEEE Trans. on Robotics and Automation*, 19:793–800.

D. Roobaert. 2001. *Pedagogical Support Vector Learning: A Pure Learning Approach to Object Recognition*. PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Sweden.

L. B. Rosenberg. 1993. Virtual fixtures: Perceptual tools for telerobotic manipulation. In *Proc. of the IEEE Annual Int. Symposium on Virtual Reality*, pages 76–82.

M. Rychetsky, S. Ortmann, and M.Glesner. 1999. Support Vector Approaches for Engine Knock Detection. In *Proc. of the Int. Joint Conference on Neural Networks*, pages 969–974.

R. D. Schraft, C. Meyer, C. Parlitz, and E. Helms. 2005. PowerMate – A safe and Intuitive Robot Assistant for Handling and Assembly Tasks. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 4085 – 4090.

M. Shimosaka, T. Mori, T. Harada, and T. Sato. 2005. Marginalized Bags of Vectors Kernels on Switching Linear Dynamics for Online Action Recognition. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 3072 – 3077.

R. Taylor, P. Jensen, L. Whitcomb, A. Barnes, R. Kumar, D. Stoianovici, P. Gupta, Z. Wang, E. deJuan, and L. Kavoussi. 1999. A Steady-Hand Robotic System for Microsurgical Augmentation. *Int. Journal of Robotics Research*, 18:1201–1210.

R. H. Taylor and D. Stoianovici. 2003. Medical Robotics in Computer-Integrated Surgery. *IEEE Trans. on Robotics and Automation*, 19:765– 781.

A. Watt. 2000. *3D Computer Graphics*, 3 edition. Addison-Wesley Publishing Ltd.

H. Woern and T. Laengle. 2000. Cooperation Between Human Beings and Robot Systems in an Industrial Environment. In *Proc. of the Int. Conf. on Mechatronics and Robotics*, pages 156–165.

L. Xie, S. Chang, A. Divakaran, and H. Sun. 2003. Unsupervised Discovery of Multilevel Statistical Video Structures Using Hierarchical Hidden Markov Models. In *Proc. of the IEEE Int. Conf. on Multimedia and Expo (ICME)*, pages III – 29 – 32.

W. Yu, R. Alqasemi, R. Dubey, and N.Pernalete. 2005. Telemanipulation Assistance Based on Motion Intention Recognition. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1121– 1126.

D. Zhang, D. Gatica-Perez, S. Bengio, I. McCowan, and G. Lathoud. 2004. Modeling Individual and Group Actions in Meetings: A Two-Layer HMM Framework. In *Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition Workshop*, pages 117– 117.

M. Zimmermann and H. Bunke. 2002. Hidden Markov Model Length Optimization for Handwriting Recognition Systems. In *Proc. of the 8th Int. Workshop on Frontiers in Handwriting Recognition*, pages 369– 374.

R. Zöllner, O. Rogalla, R. Dillmann, and M. Zöllner. 2002. Understanding Users Intention: Programming Fine Manipulation Tasks by Demonstration. In *Proc. of the Int. Conf. on Intelligent Robots and Systems*, pages 1114– 1119.