Dapeng Zhang

Switching Attention Learning

Dissertation

Institut für Informatik
Technische Fakultät
Albert-Ludwigs-Universität Freiburg

March 2011

Tag der Disputation:
12th September 2011
Dekan:
Prof. Dr. Matthias Teschner, *Albert-Ludwigs-Universität Freiburg*
Gutachter:
Prof. Dr. Bernhard Nebel, *Albert-Ludwigs-Universität Freiburg*
Prof. Dr. Martin Riedmiller, *Albert-Ludwigs-Universität Freiburg*
Prof. Dr. Kai Arras *Albert-Ludwigs-Universität Freiburg*

To whom it may concern

Acknowledgements

The idea of this thesis is from years of research that had been done since I joined in the GKI group. From then on, I have been working with a number of people who contributed in assorted ways to this research. It is a pleasure to convey my gratitude to all of them in my humble acknowledgment.

In the first place I would like to record my gratitude to Prof. Dr. Bernhard Nebel for his supervision, advice, and guidance from the very beginning of this research. He provided me support in various ways. He not only gave me this chance to gain extraordinary experiences through out the work, but also provided financial supports for me and for the students working with me. His perception to the project exceptionally inspired and enriched my growth as a student and a researcher.

I am grateful to Prof. Dr. Martin Riedmiller for his advice and supervision. I am much indebted to him for his valuable advice in our scientific discussion, his precious time on reading this thesis, and his critical comments.

I would like to thank Zhongjie Cai for helping me on the Tetris project. I am proud that I had this opportunity to work with such an exceptionally experienced software developer as him. And it is my pleasure to collaborate with Arming Hornung. Also I highly appreciated the help from Chen Kefei, Mou Wei, and Lin Zinan.

Many thanks go in particular to my colleagues. I benefited a lot from their advice and guidance. Malte Helmert and Alexander Kleiner always kindly granted me their time in answering some of my questions about CRFs and robotics. I gratefully thank Sebastian Kupferschmid for his discussion and help on this thesis. I would like to thank Ulrich Jakob for his strong support in computer hardware and software, Thilo Weigel who brought me to the world of table soccer robots. Also I would like to thank Robert Mattmüller, Christian Dornhege, and Thomas Keller for their kindly help during my working on

this project. I would also acknowledge Sun Dali, Patrick Eyerich and other colleagues for their time to play table soccer.

Many thanks go in particular to my friends,Wei Fang and Wang Yunsheng. I had a pleasant time in our discussion and I enjoyed our lunch meetings and the dry humor about scientist's life.

Words fail me to express my appreciation to my wife Liu Wei whose dedication, love and persistent confidence in me has taken the load off my shoulder. I owe her for being unselfishly let her intelligence, passions, and ambitions collide with mine.

Finally, I would like to thank everybody who made this thesis possible. And I apologize that I could not mention everybody one by one.

Abstract

Learning is an important way to acquire unknown information for both computer and human. In game competitions, human can improve themselves by combining a series of learning processes: observing and imitating others; training oneself forin a certain skill; finding a new skill and training again. In machine learning, various learning methods are developed. These methods are normally isolated for different purposes. The improvements the methods can achieve are limited to the effects of a single model, which is inevitably constrained by the prior structure of the model. Therefore, building a systematic learning mechanism is motivated, whereby multiple learners can be combined and the constrains can be relaxed. The details are addressed in the first three parts in this dissertation.

In the first part, Switching Attention Learning (SAL) is defined. The basic idea is that improving one model in the system generates more "improvement space" for the others. If the inputs of one learner come from the outputs of another, and if this relation forms a loop, the "improvement space" can be significantly enlarged by iterating over the learners in the loop. This generalizes the existing research in semi-supervised learning and boosting approaches in two aspects: It is not necessary that both labeled and unlabeled data are involved in the learning; SAL provides a framework of systematic learning, whereby the components of different layers in a system can be connected. SAL is studied in the context of game competitions. In order to test SAL, two test-beds based on table soccer and Tetris were developed.

In the second part, we explain our work on table soccer. Table soccer is a physical game that can be played by both human and robot. We built a table soccer game recorder, KiRe, so that a computer can access the game data of the human players. To explain the recorded data, a method employing Conditional Random Fields (CRFs) is initialized. Using the SAL framework, we

studied CRFs by simulation. Several learners were developed for the feature induction of CRFs. They can be used to learn a queue of CRFs models (CRF queue) over iterations. CRF queue guarantees no worse results than a single model. Besides, a novel feature reduction step was integrated in the induction process. It can significantly reduce the required time in the learning. Finally, the possibility of using this method on the real data from KiRe is discussed.

In the third part, Tetris is studied as the second test-bed. Tetris has been one of the most popular computer games for many years. The single Tetris game has been used as a test-bed by many researchers. We developed a platform for competitions. The platform is based on an open source project – KDE. Several learners were developed according to the SAL framework. Support Vector Machines and learning by imitation are employed in these learners. They can work together to build an AI player. The player has a chance to defeat the best known artificial player in the world. In addition, it can always find and imitate better players.

Switching Attention Learning is defined and discussed. We address the advantages as well as the problems of SAL, developed two test beds, and show in the experiments that SAL is a practical and systematic approach for the learning in game competitions.

Zusammenfassung

Lernen ist für Computer und Menschen gleichermaßen eine wichtige Methode, um unbekannte Informationen zu gewinnen. In spielerischen Wettbewerben können sich Menschen verbessern, indem sie eine Reihe von Lernprozessen verbinden: andere Spieler beobachten und imitieren; sich selbst in einer bestimmten Fertigkeit schulen; eine neue Fertigkeit finden und diese erneut üben. Im Maschinellen Lernen werden verschiedene Lernmethoden entwickelt. Diese Methoden werden normalerweise zu verschiedenen Zwecken isoliert. Die Verbesserungen, welche die Methoden erzielen können, sind auf die Wirkungen eines einzelnen Modells beschränkt, welches unweigerlich von der vorgegebenen Struktur des Modells eingeschränkt ist. Dies motiviert die Konstruktion eines systematischen Lernmechanismus, bei dem mehrere Lerner kombiniert und die Einschränkungen abgeschwächt werden können. Mit den Details befassen sich die ersten drei Teile der vorliegenden Dissertation.

Im ersten Teil wird der Begriff des Switching Attention Learning (SAL) definiert. Die Grundidee ist, dass Verbesserungen eines Modells im System den “Raum der Verbesserungen” der anderen Modelle vergrößern. Wenn die Eingaben eines Lerners aus den Ausgaben eines anderen Lerners kommen und diese Relation einen Zyklus bildet, dann kann der “Raum der Verbesserungen” bedeutend vergrößert werden, indem man über die Lerner in dem Zyklus iteriert. Dies verallgemeinert bestehende Forschungsarbeiten aus dem Bereich des halb-überwacht Lernens und des Boosting in zweierlei Hinsicht: Es ist nicht nötig, dass sowohl “labeled” als auch “unlabeled” Daten zum Lernen verwendet werden; SAL stellt einen Rahmen für das systematische Lernen bereit, bei dem die Komponenten unterschiedlicher Schichten in einem System verbunden werden können. SAL wird im Zusammenhang

mit spielerischen Wettbewerben untersucht. Um SAL zu testen, werden zwei Tauglichkeitstests basierend auf Tischfußball und Tetris entwickelt.

Im zweiten Teil beschreiben wir unsere Arbeit über Tischfußball. Tischfußball ist ein Spiel, das sowohl von Menschen als auch von Robotern gespielt werden kann. Wir haben ein Aufzeichnungsgerät für Tischfußball, KiRe, konstruiert, um es einem Computer zu erlauben, auf die Spieldaten der menschlichen Spieler zuzugreifen. Um die aufgezeichneten Daten zu erklären, wird eine Methode initialisiert, die Conditional Random Fields (CRFs) verwendet. Im Rahmen von SAL wurden CRFs in einer Simulation untersucht. Mehrere Lerner wurden für die Merkmasinduktion in CRFs entwickelt. Diese können verwendet werden, um eine Warteschlange von CRF-Modellen (die CRF-Queue) zu lernen. Außerdem wurde eine neue Methode zur Merkmalsreduktion in den Induktionsprozess integriert. Er kann die für das Lernen benötigte Zeit signifikant verringern. Schließlich wird die Möglichkeit diskutiert, diese Methode auf den echten Daten von KiRe einzusetzen.

Im dritten Teil wird Tetris als zweiter Tauglichkeitstest untersucht. Tetris ist seit vielen Jahren eines der beliebtesten Computerspiele. Das Ein-Spieler-Tetrisspiel wurde bereits von vielen Forschern als Tauglichkeitstest verwendet. Wir haben eine Plattform für Wettbewerbe entwickelt, welche auf einem quelloffenen Projekt aufbaut: KDE. Mehrere Lerner wurden gemäß SAL entwickelt. Sie verwenden Supportvektormaschinen und Lernen durch Imitation. Sie können zusammenwirken, um einen KI-Spieler zu konstruieren. Der dabei entstandene Spieler hat die Chance, den weltbesten bekannten künstlichen Spieler zu besiegen. Darüber hinaus kann er immer stärkere Spieler erkennen und imitieren.

Switching Attention Learning wird definiert und diskutiert. Wir gehen auf die Vorzüge ebenso ein wie auf die damit verbundenen Schwierigkeiten, entwickeln zwei Tauglichkeitstests, und zeigen in den Experimenten, dass SAL ein praktikabler und systematischer Zugang zum Lernen in spielerischen Wettbewerben ist.

Contents

Part I

Switching Attention Learning – the Idea

1

Introduction

Games are fun. They stimulate the intellect. They are an important form of entertainment in modern life. Generally, many activities are games: soccer, chess, and poker etc. Being a good player in one of these games is not very easy. For instance, many people take soccer as their hobby; even if they have been training and playing for more than ten years, most of them are still amateurs.

Some of the games can be played by computers, e.g. poker. A computer, however, plays games in a very different way. First of all, computers are machines – they have neither passion nor anger. Once programmed, a computer can sit there, repeating a routine for years without any difference and it would not complain. From this point of view, computers have abilities of repetition and accuracy much better than human beings. No one doubted that, once Deep Blue defeated the world champion in chess, it could dominate the chess world and always win.

When playing games, human beings and computers behave in diverse ways. Humans can improve their skills via learning over a fairly long period. Their performance can be changed from time to time. Computers are good at repetition. They have some learning abilities which are quite limited compared to those of human beings. Machine Learning, a research branch in artificial intelligence, deals with the learning issues of computers. Many learning methods have been developed in machine learning over the past years. Each of them is isolated for a certain issue and inevitably has its own constraints and prior structures.

This dissertation aims to relax these constraints by defining a systematic learning framework in which the existing methods can be plugged and work together. Two games are chosen as the test beds. These games can be played

by both human beings and computers for the convenience of comparison. The comparison of learning and game playing can foster interesting research.

1.1 Games in Computers

Nowadays computers can play many games and many different methods and ideas are used for games. These approaches can be classified into three categories according to the types of the games: board games, physical games, and video games. We discuss these categories one by one in this section.

In a board game, some pieces can be placed or moved (removed) on a board by players. A set of game rules define the legal movements of the game. A player can win or lose if the game reaches one of its final states. Chess is an example of such board games. Two players compete against each other in Chess . Its board consists of 8×8 grids. Each player controls 16 pieces: a king, a queen, two rooks, two knights, two bishops, and eight pawns. The player whose king is removed by the other loses the game.

For human beings, playing a board game is a brain challenge. Players need to struggle for a win. Artificial Intelligence (AI), a branch in computer science, aims at developing computer intelligence which is similar to that of human beings. Board games are widely used as test-beds in AI.

For a computer, solving a board game means the computer knows the best movement at any game states. This task is very challenging because the number of states of a game can be extremely large. For example, Chess has 10^{50} states which is far from computable by any computer. The main challenges of solving a board game are thus to find a mechanism to deal with the huge state space.

Researchers in AI developed many approaches for solving huge state space issues in board games. Herik et al. summarized the solved games and foresaw the future developments [40]. The basic idea was to build some structures or patterns to avoid visiting too many states. These structures can be an evaluation function, a heuristic function, a list of game states, a simulation of a game, and many other forms. Most of the board games have perfect state information, which simplifies the observation issues in a computer.

Physical games, the second category, are activities of human beings such as soccer, ping pang, or snooker etc. Players need to be physically involved in the games and comply with a set of rules during playing. For example, two teams of players are in soccer game. Three referees are in charge of the rules. Each team defends their own goal from being scored, while attacks the goal of the other team as much as possible.

The players need to dribble, pass, kick, and block the ball. Speed, accuracy, skills, and power are required for the players to win such a game. A human player may need to be trained for several years to improve their dribble skills, to run faster, to shoot more accurately, and to block more effectively. These improvements involve not only the brain but also the limbs, the lungs, and other parts of the body which is far beyond the necessities for playing a board game.

In most cases, a physical game is only partially observable for computers. It is different from the situation of a board game which has perfect and fully observable state information. A computer can be equipped with a camera, a laser measurement system, and other sensors to observe the environment. The measurements of the sensors are only an approximation of the real situation. This approximation is used to build a world model as accurate as possible. Also, if there are multiple players in a game, the model including the cooperation among players is challenging.

As the famous activity of scientific research, RoboCup promotes several physical games. The initiative of RoboCup is to build a team of robots that can play soccer with human beings. RoboCup fostered much research in artificial intelligence and robotics, and became very popular in recent years [46]. In this context, the works in engineering, sensor technologies, and computer science can be integrated and tested. From the previous competitions in RoboCup, we can see that the developments were amazing. However, the teams of robots are still competing against themselves and are hardly comparable to human teams so far. The situation is complicated and the problems are not east to solve.

Video games are classified into the third category. A virtual world is created in a video game. Players can enjoy gaming activities which are not possible in reality. The game is shown on a display and it is controlled by input devices such as a mouse, a keyboard and a joystick. Most of these games are fully observable because the game worlds are generated by computers. The rules of a game can be defined by its designers, human beings.

Video games have great commercial values. Solving a video game is normally not as challenging as solving board games or playing physical games. Players are required to accomplish some tasks, which are designed for fun. We will focus on the design of video games in this section.

Game design is an issue that includes platform, control, graphic, game engine, and storytelling, etc. There are different researches in these areas: the virtual roles appear to be intelligent; a game engine needs to provide interest-

ing game rules so that players will have fun in the game. These tasks can be solved by using methods in AI.

Game AI is a branch of research in which AI technologies are employed to design video games. Mateas reviewed some of these methods [52]. For example, decision tree and rule-based system can be used for decision making in a video game; a game engine can be developed by genetic algorithm or neural networks; breadth first search, deep first search, and $A*$ can be used in path-finding.

In this section, games in computers are classified and outlined. We mainly discuss the games involving both human beings and computers. Our work is based on this scenario because of the following advantages: first, computers and human beings control the game in the same way which limits and simplifies the problems; second, computers can learn from human players; and third, the learning of the computers can be compared to that of human beings.

1.2 Learning in Computers

Machine Learning (ML) is a branch of research in Computer Science. It is closely related to Artificial Intelligence (AI). The text book written by Mitchell is a good introduction to ML [54]. In AI, many approaches are from model to data. The basic method in ML is, however, to generate models or patterns from data sets – from data to models. The data can range from huge databases to single data set from one robot. Many approaches are developed in ML, three of which are the basis of this research. We will briefly introduce these directions in this section: supervised and unsupervised learning; imitation learning; and Bayesian networks.

Supervised and unsupervised learning are discussed in two examples: decision tree and expectation maximization.

Decision tree learning is a common approach in ML [14]. A decision tree can provide a classification given a number of inputs. These inputs can be represented as a vector of variables. The indicated class is decided with the values of the inputs. The labels of the classes are predefined. The task of the learning is to build a tree structure given a set of inputs and their classifications.

The tree structure resembles an upside down tree. The root node is topmost and starts the tree. The nodes are the variables. Their values create branches and the leaves are associated with classification. The input vector can be mapped to a path from the root to one of the leaves which indicates the final classification of the inputs. In the learning of a decision tree, the “tree

structure" is assumed. The tasks are to build "parent-child" relations between different values of the current nodes and its possible child nodes. A class label is assigned to each leaf.

Starting from the root, the learning method of decision tree is to pick a variable from the input vector and insert it to the tree. The core problem is an estimation function that makes one variable stand out of the candidates. This evaluation is based on "entropy" in information theory. The entropy can measure the uncertainty associated with a variable. The uncertainty in the learning encodes the possibilities that the inputs can be mapped to each class label. The difference of the entropy values is regarded as the evaluation. These two entropy values are calculated by including or excluding a variable in the tree. The difference is called information gain.

Entropy is essential in information theory for quantifying information. Shannon proposed the concept for signal processing [66] where entropy is applied for transferring data. It is widely used in data compression, channel coding, Internet, and many other fields. Entropy can be used to build several measurements for different purposes, e.g. conditional entropy, mutual information, and information gain.

Learning a decision tree requires the training data being labeled with their classes. This learning is called supervised learning. In many cases, it is hard to obtain the labeled data. Several methods were developed to learn a model from unlabeled data which is called unsupervised learning. Expectation Maximization (EM) is one of these methods.

EM algorithm was first proposed by Dempster et al. [26]. The computer iterates over an"Expectation (E)" step and a "Maximization (M)" step to compute the parameters of a certain model. The model is normally used for data classification, in which inputs are mapped to a class label. Many models can acquire their parameters via EM, e.g. Gaussian mixture and support vector machines.

In an EM algorithm, the inputs can be regarded as a data point in a multi-dimensional space. Each dimension denotes a variable in the inputs. The points that are closed to each other in the space are classified with a class label. Two labels should mark distinct subsets of the points in the space. Although concrete labels are not necessary in the learning, the number of the labels is normally required. The learning task is to find the boundaries that can best separate the subsets of the points.

This learning can be understood as maximizing the likelihood of a model given a set of unlabeled data. E and M are called iteratively in the following way: initially, the parameters of the model are chosen randomly; in the E step,

the model is then used to compute classifications of the data points; next, in the M step, the parameters of the model are computed based on the classified data; finally, E and M steps are called iteratively until the parameters are converged.

Methods based on probabilities such as maximizing likelihood are widely used in learning. Normally, these methods can be described as a graphic model. In the graph, the nodes denote the variables; and the edges denote the dependencies among these variables. There are directed graphic models, for example, hidden Markov models, and undirected graphic models, for example, conditional random fields. These models have certain power of prediction, for instance, they compute the most probable explanation of given data.

Learning probability model is challenging. Normally, each node or edge in a graphic model is associated with a parameter. The number of parameters in a graph consisting of a hundred nodes can easily reach several thousand which is hard to compute. Thus, the graphic models have to be limited by some structure and constraints.

Hidden Markov Models (HMMs) assume such constraints. HMMs consist of a chain of nodes. Each node indicates a hidden state. The state, given its predecessor, is independent to other states. This property is called the Markov property. The states are hidden because they cannot be observed directly. Normally the observations can be described as a vector. In HMMs, a parameter is a measurement of the dependencies between two states, or between a state and an observation. HMMs have been successfully applied in many sequential tasks such as speech recognition [58].

HMMs consist of a directed graphic model. Its counterpart in undirected graphic models is Conditional Random Fields (CRFs). CRFs have a similar chain structure. The main difference is HMMs are based on joint probabilities while CRFs are based on conditional probabilities. The strong dependency assumptions in HMMs are relaxed in CRFs to make CRFs more flexible. CRFs outperform HMMs in synthetic labeling tasks [45].

Many learning algorithms require considerable computational power. Imitation learning is one of the approaches for quick learning. Imitation learning is an interdisciplinary branch of artificial intelligence and biological science [13]. Typically, there are a demonstrator and a learner in the scenario. The learner observes the behaviors of the demonstrator and replicates them [50].

Imitation is employed to solve problems ranging from learning low-level actions to acquiring high-level tasks. The demonstrator and imitator could be similar or very different in the scenario. Imitation learning provides an

efficient way for quick learning, skipping the tedious self-learning process which always takes thousands of trials to acquire a proper but simple action. This advantage of imitation learning can be used to solve the typical problems faced by some robot systems. These systems normally use a huge state space where getting reinforcements from executing the actions are always too slow.

For example, a humanoid robot always has many "Degrees Of Freedom" (DOF). Searching for proper parameters for all the DOF is tremendous work [9]. Human beings have graceful actions. They could be a demonstrator. The imitation used here helps to find a good set of parameters which can consequently create a sequence of actions, being similar in shape to the demonstrator's behavior [51, 62].

In this section, several learning methods are reviewed: decision tree, information entropy, EM framework, HMMs, CRFs, and imitation learning. In all of them, some basic structures or certain forms of the data are assumed, which inevitably introduce constraints and limit the flexibility of the model.

1.3 Motivation

KiRo is a fully-automatic table soccer robot [79]. It has a reaction speed faster than a human. In 2005, KiRo could already defeat most human players, even advanced ones. There are mainly two paths for its further development: increasing the system speed until it can defeat the world champion of table soccer; or developing intelligent behaviors. We chose the latter one because it is more interesting for the research in artificial intelligence.

Both KiRo and human players control the game by turning and moving the game rods. In order to develop the intelligent behaviors for KiRo, we considered the same tasks achieved by humans. Humans can improve themselves by combining a series of learning processes: observing and imitating better players; training oneself for a certain skill; finding a new action and training again, etc. It would be wonderful if KiRo had the similar learning abilities by which it could develop some intelligent behaviors itself.

As mentioned in Section 1.2, many methods are developed in machine learning. They assume certain structures and work for specific issues. KiRo is a system that consists of layered components. For instance, the sensor data can be fused by using Kalman Filters [84]; the results can be explained by a sequence learning method such as Conditional Random Fields [45]; and the explanations can be further used for the action selection. It is hard to find a single learning method that can achieve the learning tasks for the whole

system. We thus focus on a learning paradigm that can provide a systematic learning solution and whereby different learning modules can be integrated.

A learning paradigm would lose its generality if it was only used in one system. Moreover, a good idea should work for different purposes in different systems. These motivated the development of another test-bed for the learning: Tetris. This was chosen as the second one test-bed because computers can obtain similar control power to what human beings have in a Tetris game, which provides an environment in which computers can learn from and compete against human players. In addition, Tetris is quite different from table soccer. It is a computer board game which is easy to be observed and hard to be solved.

The learning paradigm is an entrance for several interesting questions in machine learning and artificial intelligence. Our main concerns are listed here. First, can the constraints defined in the learning of each module be relaxed by other modules in a system? Second, can the paradigm generate a result more than simply adding the results of its components? Third, can a computer improve itself via bootstrapping?

1.4 Outline

This dissertation consists of five parts. In the remainder of this section, semi-supervised learning and the boosting framework are explained as the backgrounds in Chapter 2. Switching Attention Learning (SAL) is introduced and defined in Chapter 3. It generalizes the basic ideas of semi-supervised learning and the boosting framework.

In Part II, Table soccer is regarded as the first test-bed of the SAL. First, the background information is explained in Chapter 5; the developments of KiRo are briefly explained in Section 5.1.1; the definition of Conditional Random Fields is introduced in Section 5.2. The training and feature induction issues of CRFs are discussed in Section 5.2.2 and 5.2.3. Then a table soccer game recorder – KiRe , is developed in Chapter 6. It can record the games of human players. Next, we address the development of the "CRF queue" in Chapter 7. It sets a basis for implementing CRFs using SAL. Finally, the methods that bridge the gap between the recorded data and the developed CRF method are addressed in Chapter 8.

In Part III, Tetris was regarded as the second test-bed of the SAL. Different from table soccer, Tetris is a static board game in computers. In Chapter 10, the "state of the art" research in Tetris is addressed. Bandit based Monte Carlo planning is used as a method for the game competitions in Chapter 11. It

shows how a computer can play the game. Learning by imitation is employed in the SAL framework for playing Tetris in Chapter 12.

In Part IV, two applications are developed which are the joint works with other students. In Chapter 13, a game controller based on multiple sensors is constructed. There is a discussion in human computer interface. In Chapter 14, an automatic referee of table soccer is implemented based on KiRe.

In Part V, we draw the conclusion and discuss possible future avenues for work. This dissertation is based on the following publications.

- Dapeng Zhang and Bernhard Nebel.**Recording and Segmenting Table Soccer Games – Initial Results.** *In Proceedings of the 1st International Symposium on Skill Science 2007* (ISSS 2007), page. 193-195.
- Dapeng Zhang, Bernhard Nebel and Armin Hornung. **Switching Attention Learning - A Paradigm for Introspection and Incremental Learning**. *In Proceedings of Fifth International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2008)*, page 99-104. Linz, Austria.
- Dapeng Zhang and Armin Hornung. **A Table Soccer Game Recorder**. *In Video Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008)*. Nice, France.
- Armin Hornung and Dapeng Zhang. **On-Line Detection of Rule Violations in Table Soccer**. *Proceedings of the 31st Annual German Conference on AI (KI 2008)*, page 217-224. Springer-Verlag.
- Dapeng Zhang, Cai Zhongjie, Chen Kefei and Bernhard Nebel. **A Game Controller Based on Multiple Sensors**. *In Proceedings of the Fifth International Conference on Advances in Computer Entertainment Technology (ACE 2009)*
- Dapeng Zhang, Cai Zhongjie and Bernhard Nebel. **Playing Tetris Using Learning by Imitation**. *In Proceedings of the 11th annual European Conference on Simulation and AI in Computer Games (GAMEON 2010)*.
- Dapeng Zhang and Bernhard Nebel. **Feature Induction of Linear-Chain Conditional Random Fields - A Study Based on a Simulation**. *In Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART 2011)*.
- Cai Zhongjie Dapeng Zhang, and Bernhard Nebel. **Playing Tetris Using Bandit-Based Monte-Carlo Planning**. *AISB 2011 Symposium: AI & Games*.

2

Preliminaries

Switching attention learning is a paradigm to integrate multiple learning parts. In the literature, learning from the labeled and unlabeled data can be combined in semi-supervised learning. Weak classifiers can be boosted into a strong one in boosting framework. Both of the branches of research handle issues of integrating and combining multiple learning parts. We briefly review them as the preliminaries of the SAL in Section 2.1 and 2.2.

2.1 Semi-Supervised Learning

Supervised and unsupervised learning were introduced in Section 1.2. There is a branch of research in which they are combined: Semi-Supervised Learning (SSL). Zhu et al. wrote a book for a general introduction to SSL [92]. Generally, SSL is an approach for data classification. Both unlabeled and labeled data are used in SSL. The unlabeled data is to improve the supervised learning which is based on the labeled data. Several concrete methods were developed in the context.

Here we denote the instances in the data set as a vector of features $x = (v_1, v_2, ..., v_D)$ where D is the dimension of the vector. $Y = y_1, y_2, ..., y_n$ is a set of labels, where n is the number of the labels. Data classification is to find a mapping function f which defines a correspondence between the instances of the data and their labels $f : x \rightarrow y, y \in Y$. In supervised learning, a set of labeled data $\{(x_1, y_1), (x_2, y_2), ..., (x_I, y_I)\}$ are given as the training set where I is the number of the instances in the set. Based on the given data, f is computed using a learning algorithm. In unsupervised learning, e.g. the EM algorithm in Section 1.2, the training set is $\{x_1, x_2, ..., x_I\}$. The number of the labels n is required as additional information. In semi-supervised learning, both labeled data $\{(x_1, y_1), (x_2, y_2), ..., (x_I, y_I)\}$ and

unlabeled data $\{x_{I+1}, x_{I+2}, ..., x_{I+J}\}$ are given, where J is the number of the unlabeled data.

In reality, annotating the data (x_i, y_i) is a very tedious job. In SSL, J is normally much bigger than I. SSL is motivated because it provides a solution to reduce the amount of work in data annotation which is expensive in practice.

An instance x can be regarded as a point in a space of D dimensions. A label y marks a cluster of the instances. SSL assumes a smooth property: if x and x' are close to each other in the instance space, their labels, y and y', are also close to each other. Based on this assumption, the unlabeled data can improve the accuracy of the boundaries among the clusters. This is the basic principle of semi-supervised learning. Several methods were developed in SSL.

Self-training or bootstrapping [85] is one of the methods in SSL. From the labeled data, a model is first trained by using supervised learning. In the second step, this model is used to predict the class labels over a set of the unlabeled data. And thirdly, the most confident predictions and their corresponding instances are added into the training set. Consequently, the model can be trained again based on the bigger training set. By iterating over these training steps, the model is bootstrapped by only a small amount of the labeled data.

In this approach, the predictions, which normally contain some mislabeled data, are regarded as labeled data in the supervised learning. If there is no special method to deal with the mislabeled data, the errors will be considerred as the correct data and they will propagate in the training. The behavior of the models is interesting because it takes its own outputs as the inputs in the next iteration.

Co-training is another method in SSL. It was first proposed by Blum and Mitchell [11]. Co-training has a significant contribution to the area. The idea is to make intra-learning between two models. One is built on a subset of features. The other is based on another different subset.

An example of the co-training is illustrated in Figure 2.1. Each of the data instances has 6 features. A labeled data set is denoted by an ellipse. The training set is divided into two subsets. In each subset, three features are considered. *Learner1* is based on the features $\{F_1, F_2, F_3\}$. *Model1* is trained by using supervised learning. An unlabeled data set, D_1, which is denoted by a circle, is fed to the model M_1 for the predictions. The most confident predictions and their data instances are added into the training sets of another learner, *Learner2*, which is based on the features $\{F_4, F_5, F_6\}$. In the same

way, *Learner2* can also be used to enlarge the training set of *Learner1* and improve the performance of M_1. In other words, M_1 is finally improved by its own predictions.

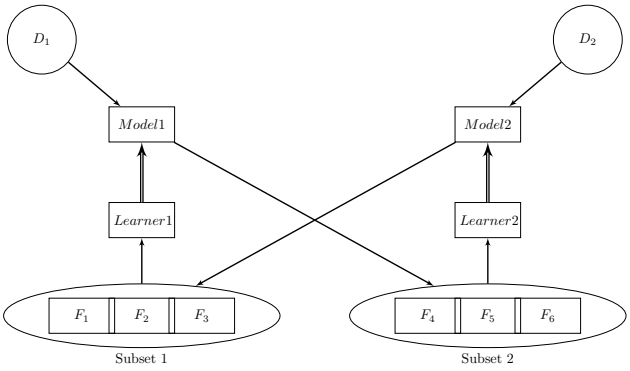


Fig. 2.1. An example of Co-Training

Blum and Mitschell not only proposed the basic idea of co-training but also analyzed the co-training in the framework of Probably Approximately Correct (PAC) learning [11]. PAC learning was proposed by Valiant [74]. It is a framework to analyze and classify the difficulties of the learning problems. In PAC, the required number of training instances is related to the probability of learning a model within a limited error rate. Co-training is a framework that can accommodate different methods and algorithms.

Multi-view learning, first proposed by Virginia [76], extends the idea of co-training. In the scenario, there are multiple learners. These learners interact with each other to obtain an agreement on the training data. The form of multi-view learning is more flexible. Different methods can be integrated into one view in the learning. This integration is much further than choosing the subsets of the features in the co-training. For example, a decision tree can be grown according to the predictions of a support vector machine. This flexibility inevitably introduces risks to the learning process [68].

As we mentioned before, SSL assumes a smooth property. It is intuitive to represent this smooth property as a surface in a multi-dimensional space. Graphic based methods are based on this notation, in which each x is denoted by a point in the space. The similarity between two points is modeled by an edge. A target function, e.g. harmonic function [91], is designed for the evaluation of the smoothness, in which the assignments of y are encoded.

In Semi-Supervised Support Vector Machines (S^3VMs), the tasks are solved by an idea from another aspect. Instead of searching for the smoothness, the gap or margin among the clusters is maximized. These clusters denote different values of y. In SVMs, only labeled data are used in the training. In S^3VMs, both labeled and unlabeled data are involved which introduces an extra problem – a non-convex optimization. Chappelle et al. wrote an overview on the approaches in this direction [21].

In general, semi-supervised learning deals with data classification issues using both labeled and unlabeled data. The methods in the area can be roughly classified into two categories. The first one is based on the interaction of multiple learners, e.g. co-training and multi-view learning. In the second category, the methods deal with both labeled and unlabeled data, for example, graphic based methods and S^3VMs. The methods in first category are particularly related to our research.

2.2 Boosting Framework

Boosting framework, first proposed by Kearns et al. in 1988 [42], is a popular method in machine learning. It can "boost" a number of weak classifiers from the training data and combine them to obtain a strong classifier which has high accuracy. The weak classifiers can have different forms which makes the framework flexible. There are several different "boosting" approaches. Schapire wrote an overview on the boosting approaches [64]. We will focus on the AdaBoost in this section. It is one of the most important methods in boosting which was also analyzed in PAC learning model [65].

AdaBoost was first proposed by Freund et al. in 1995 [34]. Two sets of parameters are learned in AdaBoost. The first is a set of weights, each of which is associated with an instance in the training set. A weak classifier can be learned based on the weighted instances. Here the weights indicate the importance of the instance to the learned classifier. In each iteration, the weights are updated so that the algorithm focuses on a different subset of the training data. A series of the weak classifiers is learned over the iterations. The second parameter set measures the contribution of each weak classifier. The final decision is simply the sum of the weighted decision of each classifier.

Figure 2.2 shows an example of AdaBoost. The learner generates the fist parameter set P_1, which is applied to each instance in the training set. The parameter set makes the learner focus on a subset of the training data. We illustrate this subset as D_1 in the figure which is intuitive. In principle, there is no clear boundary between D_1 and other data. Then the learner generates

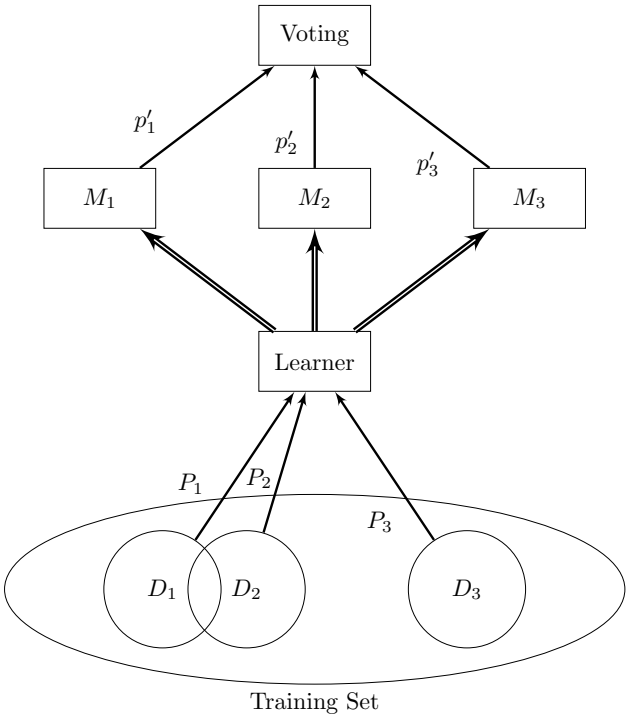


Fig. 2.2. An example of AdaBoosting

M_1 which is a weak classifier. In addition, a weight parameter p'_1 is chosen to be associated with M_1 in this iteration. In the same way, M_2 and M_3 can be learned one after another in the next two iterations. The three models can be combined in a voting mechanism for the prediction. The prediction of each model is multiplied with the corresponding weight factor in $\{p'_1, p'_2, p'_3\}$. The resulted values are added together for the final prediction.

AdaBoost has been applied in a wide range of applications, for instance, spam detection, image retrieval, medical diagnosis, and drug discovery. The idea of boosting can be combined with semi-supervised learning for both labeled and unlabeled data [47]. It is an interesting approach to this dissertation because it provides a way to learn multiple models and combine these models for the prediction.

3

The Definition of Switching Attention Learning

Attention plays a very important role during the learning processes of human beings. For example, students in university take several courses in the morning. The subjects could be dependent or independent. The understanding of physics requires some background in mathematics. Studying English is not directly related to learning biology. In any situation, the students can develop their knowledge by switching their attention from one subject to another. Switching attention can be observed not only in complex tasks as in the example above but also in some much simpler ones such as learning a one-second sport action. For instance, a human player tries to improve his dart for six months. M. Suwa found that together with a measurable improvement of his overall performance, the player switched his attention to different parts of his body, e.g. the waist, the elbow, or the fingers. Based on the "switched attention", the key mechanism of the improvement is explained as the so-called "Meta-Cognitive Verbalization" [69].

Human beings have the ability of learning to improve their skills. When facing a learning task, such as playing darts, people can acquire the necessary skills by repeating the following three steps: finding a weakness, overcoming it, and switching attention to another weakness. Iterating over the steps can be regarded as an incremental-learning process with introspection. In artificial intelligence (AI), there exist many learning approaches which have are very useful and powerful.

SSL and boosting approaches are towards solving problems in data classification which is an identified research issue. In general, a clue to attack such an issue is to first develop a set of standard benchmarks; then to develop some methods for the problems and compare the methods based on these benchmarks. This is called a research-oriented development.

Our research is application-oriented. As mentioned in Section 1.3, our research was first motivated by developing KiRo further. Instead of focusing on a specific issue, e.g. data classification, we investigated a series of methods which were necessary and proper for the development of the application. The ideas can be tested in the application and then be generalized and applied in other similar systems.

Many systems have a layered structure. The problems in different layers can hardly be described or modeled using a single method. For example, a table soccer robot has three sub-systems: world model, action control, and action selection. To make the robot play table soccer well, we need to recognize the position of the ball, have a number of basic actions, select from these actions, and be adaptive to different players. These require different methods working on different layers. The methods can hardly be related to a certain learning issue, e.g. data classification. We define switching attention learning for the development of a learning system. It is application-oriented. Instead of performing a deep research in a certain direction, SAL is designed to integrate a wide range of different methods.

In short, *Switching Attention Learning* (SAL) is a paradigm in which multiple learners can be plugged. The inputs of one learner come from the outputs of the others.

To explain in detail, a SAL system consists of the following four elements:

- a set of system goals, $\mathbf{G}$;
- a set of models, $\mathbf{M}$, cooperating for $\mathbf{G}$;
- a set of data and structures $\mathbf{D}$, being input and output of the models in $\mathbf{M}$;
- a set of learners $\mathbf{L}$, which can improve the models in $\mathbf{M}$ towards G.

A model m is an active model if there is a learner l_m improving m, where $m \in \mathbf{M}$, $l_m \in \mathbf{L}$. The input of the model "m" is denoted as I_m and its output as O_m, where $I_m, O_m \subset \mathbf{D}$. So far, an active model can be defined by a 4-tuple, (m, l_m, I_m, O_m). If the set of the active models is $\mathbf{M_a}$, we have $\mathbf{M_a} \subseteq \mathbf{M}$.

We use "d" to denote an element in $\mathbf{D}$. d is an active element if $d \in I_{m_1}$ and $d \in O_{m_2}$ where $m_1, m_2 \in \mathbf{M_a}$ and $m_1 \neq m_2$. With "$\mathbf{D_a}$" denoting the set of all active elements, d_i and d_j are "connected" by m if $d_i \in I_m$ and $d_j \in O_m$, where $d_i, d_j \in \mathbf{D_a}$, $m \in \mathbf{M_a}$. There is a path from d_1 to d_n if $\forall d_k, d_{k+1} \in \{d_1, d_2, ..., d_n\}$, d_k and d_{k+1} are connected. **A system complies with the switching attention learning paradigm if there is a path from an active element to itself**.

The definition of SAL limits our domain to closed-loop multiple learner systems. *Closed-loop* is a concept which is widely used in control theory and

artificial neural networks. The idea is to notify a system with feedback of its own performance. *Closed-loop* is important for the SAL paradigm because the learning process of human beings, which inspires the initial idea of SAL, appears to be closed-loop. We believe it can gain advantages of incremental learning.

We can distinguish a SAL system from others by its definition. There are similar learning systems in which different learners are involved. For example M. Fox and her colleagues developed an approach [33] where the structure of Hidden Markov Models (HMMs) can be learned by a learner using Kohonen networks. The parameters of the HMMs are computed by another learner using Expectation Maximization framework. The approach is different from ours because the "active elements" in the system do not form a loop.

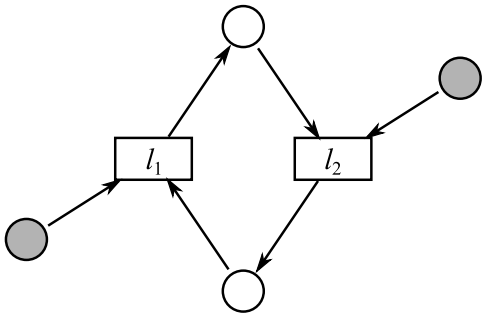


Fig. 3.1. An Example with Two Learners

SAL requires at least two learners. Figure 3.1 shows an example. The rectangles are learners. The circles are their inputs and outputs. The white nodes are the *active elements*. They can be regarded as the medium of the improvement. If we ignore the two gray nodes and their peripheral arrow lines, the remaining nodes and lines which form a loop show an example of the smallest possible SAL.

SAL provides a flexible context in which a specific real-world problem can be divided into sub-problems, each of which can be solved in an individual model. The development process can pass three stages. First, the sub-problems are solved independently with the models being defined with their input and output. In this stage, we can expect an open-loop system, in which the models are independent or they can do only a few interactions but not incrementally. Then we focus on the elements which can bridge different models, adjusting them so that they can represent both input and output. Fig-

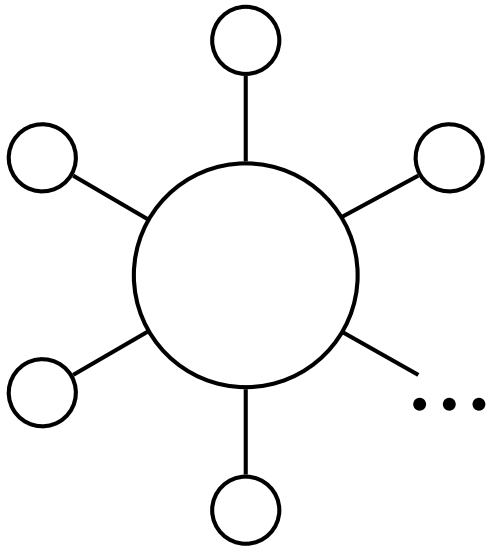


Fig. 3.2. Active Elements

ure 3.2 shows the development map of this stage. The big circle at the center can be regarded as the closed-loop learning. We need to find out the *active elements* in the system and develop learners for them. Finally, the dependency among the models and learners are analyzed. We use a dependency map to show these relations. Figure 3.3 shows the dependency map of the two-learner example. We can design an attention model according to the topology of the dependency map. In the example, the learners should be simply performed one after another. We believe designing a good attention model can accelerate the learning process in a more complex context.

SAL provides a framework for exploring the relations among different models and learners. The essential characteristic of SAL is that the structure based on the models and learners produces the power of introspection and incremental learning. SAL is supposed to be used in scenarios where achieving final goals requires a few intelligent components. Each component uses distinct representation and algorithms and solves problems from different points of view. It also provides a platform where different solutions for the same problem can be compared and combined. One of the possible applications is the table soccer robots [79] in Part II. The robot should not only play against human beings and be adaptive to different human players but also classify the behaviors of the players and learn from them. SAL can also be used in much smaller applications, e.g. in Tetris in Part III.

Fig. 3.3. Dependency Map

SAL generalizes the ideas of SSL and boosting. In AdaBoost [64], once a weak classifier is boosted in one iteration, a new set of parameters are computed for the next iteration. If the behavior of each iteration is regarded as executing a learner, AdaBoost complies with SAL because the training data are filtered by the distribution parameters and passed to the next iteration (learner). Co-training, mentioned in Section 2.1, complies with SAL because each of the learners takes the outputs of the other as its inputs, as shown in Figure 2.1. In Figure 3.1, an example of SAL with two learners is shown which has exactly the same meanings as Figure 2.1.

SAL is different from AdaBoost and co-training mainly in three aspects. First, SAL can deal with issues of a system, especially the real system with a layered structure. Second, SAL can be used to describe AdaBoosting or co-training; it does not assume a context of labeled and unlabeled data or weak classifiers. Third, SAL is application-oriented; we did not yet find a way to analyze SAL using PAC learnable, but its practical values can be shown in the experiments.

So far, we defined the basic concepts of SAL, showed some possible tools in the developing process, illustrated some application domains, and compared it with other approaches. There are many problems remaining. One problem is that improving a single learner in the system does not guarantee a better performance of the whole system. Another problem is that the errors generated by each learner are possibly accumulated in the system which will cause later mistakes. SAL is very flexible in practice which makes the analysis, e.g. PAC learnable, very difficult. We expect that the difficulties should be overcome by designing an evaluation model. However, developing many models and structures normally requires a lot of time which makes it difficult to try or prove the concepts in SAL. Nevertheless, we believe the problems will be met and can be solved on-the-fly.

Part II

Table Soccer

4

Outline

In this part, we first introduce the developments of the table soccer robots and a probability model that can be applied to explain the data of table soccer. This information is the preliminary of this part. Then, we address the hardware and software extensions of the table soccer robots which create a platform for the learning research. Next, we study the conditional random fields in a simulation in which SAL and CRFs can be combined and CRF queues are defined. Finally, we discuss the possiblility of explaining the data of table soccer using CRFs.

5

Preliminaries

Table soccer (or foosball) is a popular game in which two teams play against each other. The first table soccer robot, KiRo, was built in Universtiy of Freiburg in 2002. It has been a research topic for over ten years [79]. KiRo has been a very interesting application [56], which was developed to a commercial available product [78] in 2005. There is a series of developments with KiRo including the vision system [82], action control[89], action selection [71], and being adaptive to human opponents, etc. [81] [80]. Our research is motivated by developing KiRo further. The above works are briefly introduced as the basis of the further developments.

Playing table soccer can be described as performing a sequence of actions, e.g. passing, dribbling, and attacking. Considitional random fields are a probability model for sequential prediction issues. It was first proposed by Lafferty et al. in 2001 [45]. There are several ways to obtain the parameters of CRFs: iterative scaling [45], Quasi-Newton method [32], and stochastic gradient approaches [77]. The features of CRFs can be induced automatically. McCallum proposed the first feature induction algorithm [53]. Dietterich et al. employed the boosting framework to solve this problem [27]. Liao et al. developed the boosting approach further for the feature induction of CRFs [48]. CRFs, combined with switching attention learning, are employed for the sequential learning. The basic methods in the context are introduced as preliminaries.

5.1 Table Soccer Robots

Two teams compete against each other in table soccer games. Each team has 4 game rods: attacker, midfield, defender, and goalkeeper. The playing figures are fixed on the rods. These rods have respectively 3, 5, 2, and 1 playing

figures. The ball is moving on a playing surface, which is a field of $1200mm$ in length and $680mm$ in width. There are two goals at the ends of the playing field. The rods have two degrees of freedom, moving (sliding) and turning. Each team controls the rods to play the game, attacking the goal of the other team, and defending their own goal from being attacked.

5.1.1 KiRo

Table soccer robot KiRo can control one side of the game to play with a human team controlling the other side, as shown in Figure 5.1. A camera is installed above the playing field. Four control units are mounted on one side of the table. The camera and the control units are connected to a computer so that the computer can observe and control the game. In the case of KiRo, the blue side is controlled by the computer.

Fig. 5.1. Robot KiRo

Figure 5.2 illustrates the control unit. Two DC motors are used to control the rod. The first motor is in charge of turning, and the second one is in charge of sliding. The first motor slides with the rod. The motor is controlled by a motor controller which provides an interface between the analog motor and the digital computer. By sending ascii commands via a serial connection to the controller, the motor can be set to move with a certain speed, move to a specific position, or hold the position. Motor controller can provide the encoded position of the motor which is the position or angle of the rod in this case.

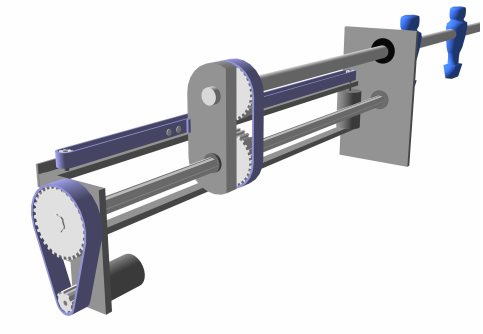


Fig. 5.2. The mechanical system of a game rod [79]

Figure 5.3 shows the view of the overhead camera. In its vision, the playing field is dark green. The field lines and the rods are white. The playing figures are red or blue. We use a special yellow ball in KiRo so that it can be easily recognized by color.

A color classification approach is employed for the observation [79]. A pixel and its neighbors are merged into one region if they have similar values in RGB color. The white pixels, which are field lines and the rods, should not change a lot from one snapshot to another. The playing field is located in the camera view according to the white pixels and the domain knowledge of the field line. Within the playing field, the yellow region with the closest size to the ball can be used to estimate the position of the ball.

The color classification algorithm depends strongly on the lighting condition. If KiRo is moved from one place to another, the lighting may change significantly. This change will affect the recognition of the ball. Solving this problem requires an adaptive vision of KiRo. Two methods are developed to automatically adjust the parameters of the color classification algorithm [82],

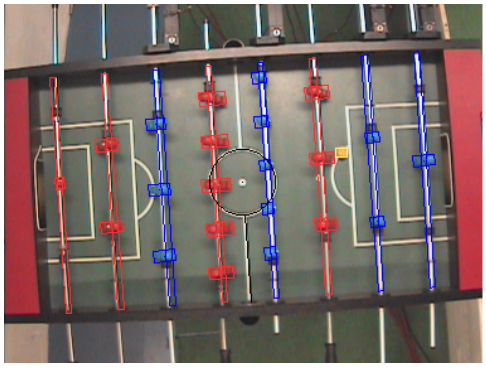


Fig. 5.3. The view of overhead camera [82]

so that KiRo can be adaptive to different lighting conditions. The first one is based on the overall percentage of a particular color in the camera view. The second one is developed by considering the shape of the scans along the length direction of the table. The histogram of the YUV values reveals the lighting conditions.

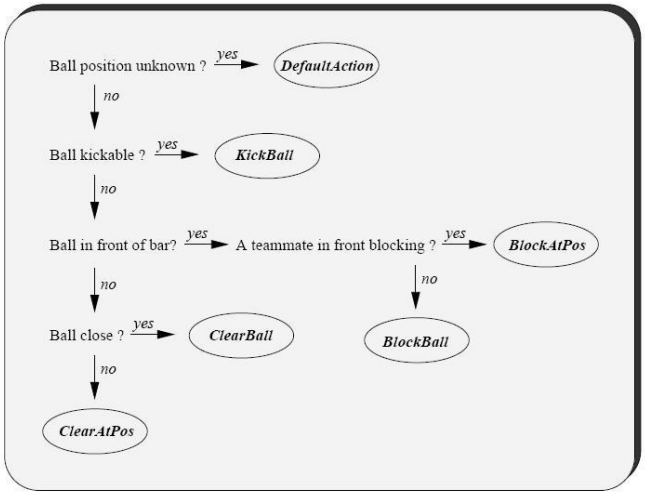


Fig. 5.4. The strategy of KiRo [79]

The strategy of KiRo in games can be described by a decision tree, shown in Figure 5.4. When the opponents control the ball, KiRo blocks the possible attacks according to a static pattern and stays in defense mode. The pattern is

acquired in the experiments which is very effective for defending. When the ball is at a kick-able position, KiRo will kick it out. In KiRo, the computer grabs images from the camera 50 times per second. The motors in the control units are powerful. Thus, the reaction speed of the system is faster than most of the human players. This makes KiRo win most of the games against human players [79].

Kicking all the time is, however, not an interesting strategy. Tacke et al. developed an action selection algorithm using decision theoretic planning [71]. In this approach, the action selection was done by searching in the possible future states. These states can be computed online in a simulator. In the experiments it outperformed the decision tree in the simulation. But with the real table, the decision tree is better.

Another approach for the action selection is based on Markove Decision Processes (MDPs) [86]. The playing fields are divided into regions. The regions are combined with the playing figures. The states of the MDPs are defined by these regions. A series of actions are hand-coded in which complex actions such as slide-kick are implemented. MDPs can be learned via reinforcement learning. The approach worked well in a simplified scenario with the real table. However, for the whole game the learning process requires a number of trials which are too tedious to be finished.

The complex actions, e.g. slide-kick, can also be acquired automatically by using imitation learning [89]. Both human players and the computer control the game via turning and moving the game rods. The actions of the human players can thus be directly mapped to the actions of the robot. An algorithm is created to clone the actions of human automatically. The "lock" and "slide-kick" actions can be learned with about 20 trials.

5.1.2 KiSi

KiRo supports physical games. The experiments of some research, e.g. reinforcement learning, are not easy to be done in physical games in which the game cannot be fully controlled. In order to make game fully controllable, we developed a simulation. The GUI of the simulation is show in Figure 5.5.

The software is constructed using a 3D library which is an implementation of openGL. A virtual game table is defined as well as the rods, playing figures, and the ball. They have exactly the same dimensions as the real ones. The force, e.g. the gravity, collisions and the friction, and the movements are simulated by using a physical engine – *open dynamic engine*. The software provides a virtual environment which is quite similar to that of a real table.

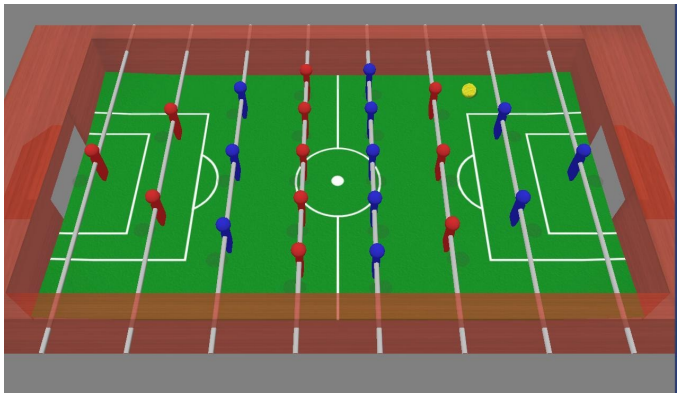


Fig. 5.5. The simulation of Table Soccer

The program for KiRo in the computer can also control the game in the simulation.

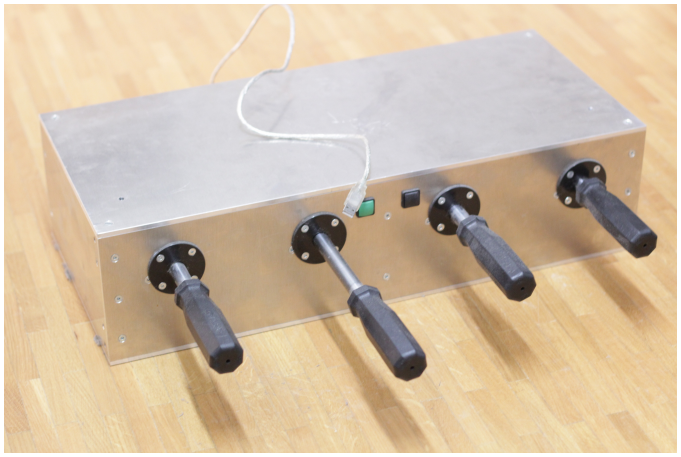


Fig. 5.6. The toy controller of table soccer

The games in the simulation are fully observable. The computer can access and define all data of the game world at any time, e.g. the speed of the ball, the position of a particular playing figure. The learning can thus be done automatically. The ball can be reset at the start postion after a trial. The scores and the behaviors of the players can be recorded. The simulation can also be used for predicting what will happen in a physical game. For instance, in the

decision theoretic planning [71], the simulation is used to predict the situation in the future.

A toy controller is developed for human to control the virtual game, as shown in Figure 5.6. The controller can be connected to the computer via a serial or USB connection. Four rods are built in the controller as the human-computer interface so that the virtual game can also be played by human players. Consequently, human and computer (robot) can compete against each other in virtual environments.

5.1.3 Star-Kick

KiRo is the first robot that can play table soccer in competitive level with human players. In an evaluation, KiRo can win about 75% of all games [79] which challenges the participants and brings amusement. This potential commercial value first pushed KiRo to a patent [56]. Then, by the cooperation between the University of Freiburg and Gauselmann AG., it was developed to the second generation in 2005: Star-Kick [78],

Fig. 5.7. Robot Star-Kick

Figure 5.7 is a picture of Star-Kick. The company developed it into a game machine which can stay alone in arcades and provide service by inserting coins. The improvements are from three aspects: the appearance, the control system, and the vision system.

Star-Kick has a friendly user interface, as shown in Figure 5.7. The monitor on the top shows the scores and the state of the machine. There are three buttons in the control panel which provide access to three different game levels and the game tips. The playing field and the game rods are covered by a piece of glass to prevent the players from being hurt by the playing figures. The ball can be thrown in from the middle and come out from the left or the right side of the control panel after a goal.

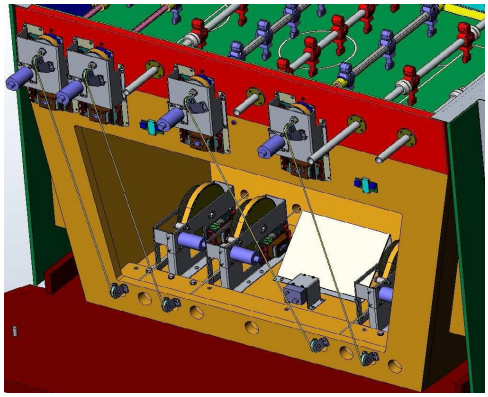


Fig. 5.8. The Control System of Star-Kick [78]

Figure 5.8 shows the inside of Star-Kick from behind. The four motors at the upper part control the turning of the rods and the four at the lower part control the moving. The movements are transferred via the wheel and the rope which is more accurate and robust than the original slide system in KiRo. By this design, the control system is finally encased inside the table.

The camera of Star-Kick is also in the body, shooting at a mirror. Figure 5.9 is an illustration to the vision system. The playing surface is semi-transparent. Two rows of LED emit infrared ray so that the lower part of the ball has a higher temperature than other parts in the vision. The camera can thus observe the lighted up part of the ball through the playing surface. The estimated position of the ball can thus be calculated according to view of the camera.

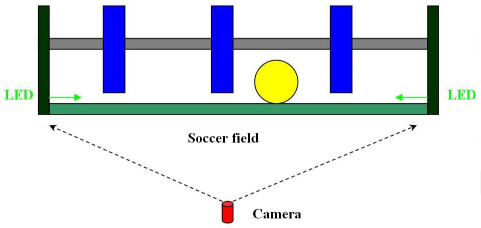


Fig. 5.9. The vision of Star-Kick

With the improved hardware, Star-Kick, as a product available on the market, can win about 90% of the games against human players [78].

5.2 Conditional Random Fields

Labeling sequential data is a chore in many domains. In natural language processing, a sentence (a sequence of words) needs to be "understood" by a computer. An important task is to label the phrases with e.g. noun, verb, or preposition in the sentences, mapping the segments of the words to the labels. In robotics, agents are equipped with sensors to acquire the measurements of the surroundings. The task of labeling is to identify the states according to the temporal sensor data. The states are normally encoded in a vector of variables with discrete values. Similar applications can be found in image processing and in computational genetics.

Researchers have developed several approaches for sequential labeling tasks. Hidden Markov Models (HMMs), for example, is a well-developed generative model suitable for such a task. The inference of HMMs is based upon joint probabilities. There is a strong independence assumption in HMMs: the observations are independent given the label. Although this assumption can hardly be satisfied in real applications, HMMs have been successfully applied in many sequential labeling tasks, e.g. in speech recognition [58].

Compared to HMMs, Conditional Random Fields (CRFs) has a shorter history. It was proposed by Lafferty et al. in 2001 [45] and gained popularity quickly. CRFs is a discriminative model based on conditioned probabilities. It has the advantage that the strong independence assumptions are relaxed. In CRFs, a hidden label is globally conditioned on all the observations in the sequence. CRFs outperformed HMMs in the experiments on the benchmarks in natural language processing [45].

This section is organized as follows. First we introduce the configuration of the sequential data in subsection 5.2.1. Then the definition of CRFs is given in 5.2.2. CRFs consists of a set of feature functions F and their weights . We address how to estimate the weights. Finally, in Section 5.2.3 we explain how F is iteratively induced.

5.2.1 The sequential Data

A game recorder was developed to record table soccer games of human [88]. The data are collected from 14 sensors which are mounted on a regular game table. They measure the position and angle of each game rod as well as the position of the game ball. The Frequency of the recorder is about $200Hz$. The sensor data are transferred to 56 Boolean variables via a discretization method. The labeling task is to identify the skills of human.

In this work, we define the data by using a typical notation in data classification. The sequential data has the form (X, Y) where X is an observation sequence $(B_1, B_2, ..., B_I)$ and Y is the state sequence. $Y = (y_1, y_2, ..., y_I)$ where I is the length of the sequence. At each state y_i a corresponding B_i can be observed, which is a vector of Boolean variables. $B_i = (b_{i,1}, b_{i,2}, ..., b_{i,C})$ where C is the number of the variables. As we mentioned above, C is 56 here.

5.2.2 Training CRFs

Conditional Random Fields is an undirected graphic model in the exponential family. The clique decomposition of CRFs supports the inference of the distributes in an arbitrary graph structure. We focus on linear-chain structure in this dissertation. In CRFs, the probabilities of a sequence of labels Y given the observations X are defined in the following equation.

$$p(Y|X) = \frac{exp(\sum_{i=1}^{I} \Theta \cdot F(y_{i-1}, y_i, X))}{Z(X)} \tag{5.1}$$

In this equation, F is a vector of feature functions $(f_1, f_2, ..., f_K)$, Θ is a vector of the weights $(\theta_1, \theta_2, ..., \theta_K)$. $\Theta \cdot F$ is the inner product of the vectors.

$$\Theta \cdot F = \theta_1 f_1 + \theta_2 f_2 + ... + \theta_K f_K \tag{5.2}$$

exp is the exponential function. Each f_k, $k \in \{1, ..., K\}$ is a conjunction test involving one or more elements in $\{y_{i-1}, y_i\}$ and X. For example, a feature function f_k can be defined as below.

$$f_k(y_{i-1}, y_i, X) = \begin{cases} 1 \textbf{ if } \; y_i = 3 \wedge b_{i-1,1} = 0 \wedge b_{i+2,4} = 1 \\ 0 \textbf{ otherwise} \end{cases} \tag{5.3}$$

$b_{o,c}$ denotes a certain observation primitive at y_i where o is the position, $c \in \{1, ..., C\}$. The value of $b_{o,c}$ is given in the observation X.

The upper part of the fraction in Equation 5.1 is the accumulated $\Theta \cdot F$ along the path Y which can be rewritten as a function of X and Y.

$$p'(X, Y) = exp(\sum_{i=1}^{I} \Theta \cdot F(y_{i-1}, y_i, X)); \tag{5.4}$$

In this equation, y_i succeeds y_{i-1} in the sequence Y. The lower part $Z(X)$ in Equation 5.1 is a normalization factor. It is the sum of the weighted feature functions of all the possible permutations of the paths which depends only on the observation X.

$$Z(X) = \sum_{Y} p'(X, Y) \tag{5.5}$$

Lafferty et al. used matrices to compute $p(Y|X)$ [45]. The special *start* and *stop* labels are added to Y which makes the notation of the matrices simpler. Adding these labels increase the estimation of the normalization $Z(X)$ because more path permutations involving *start* and *stop* are taken into consideration. In our implementation, they are removed.

Suppose (X, Y) is an annotated sequence. For a y in the sequence Y, y has $\mathbb{Y}$ possible values. For each position $i > 1$ in the sequence, a matrix $M_i(X)$ of $\mathbb{Y} \times \mathbb{Y}$ is defined. Each cell $m_{i,y',y}$ in the matrix at the entry (y', y) is computed via the following equation.

$$m_{i,y',y}(X) = exp(\Theta \cdot F(y', y, X)) \tag{5.6}$$

We need a $\mathbb{Y} \times 1$ end vector V_e of the form $(1, 1, ..., 1)^\top$ and a $1 \times \mathbb{Y}$ start vector $V_s = (v_1, v_2, ..., v_{\mathbb{Y}})$ where each v_j, $j \in \{1, 2, ..., \mathbb{Y}\}$ is defined as follows.

$$v_j(X) = exp(\Theta \cdot F(\perp, j, X)) \tag{5.7}$$

The conditional probability of the annotated sequence (X, Y) is given in the following equation.

$$p(Y|X) = \frac{v_{y_1} \prod_{i=2}^{I} m_{i,y_{i-1},y_i}}{V_s \prod_{i=2}^{I} M_i(X) V_e} \tag{5.8}$$

CRFs is usually used to compute the most possible explanation (labels) of an observation sequence which can be done via Viterbi algorithm and dynamic programming.

The parameters Θ of CRFs can be estimated by a training process in which F is assumed to be known. Given the training data $D = (\mathbf{X}, \mathbf{Y})$ where $\mathbf{X} = \{X_1, X_2, ..., X_N\}$, and $\mathbf{Y} = \{Y_1, Y_2, ..., Y_N\}$, the training algorithm maximizes the likelihood of the CRF model.

$$\Theta^* = \arg\max_{\Theta} \sum_{n=1}^{N} p(Y_n|X_n) \tag{5.9}$$

There are chiefly two types of approaches to train CRFs. One is based on iterative scaling and Newton's methods. The other uses stochastic gradient methods. In CRF++ [1], a quasi-Newton approach using Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) update was implemented. We choose the stochastic gradient approach [77] in our implementation. For $\theta_k \in \Theta$, the gain $g(\theta_k)$ can be estimated as follows.

$$g(\theta_k) = \frac{\theta_k}{\delta^2} - \sum_{n=1}^{N_b} \sum_{i=1}^{I_n} (f_k(y_{i-1}, y_i, X_n) - \mathbb{E}_{\Theta} f_k) \tag{5.10}$$

In Equation 5.10, δ is a constant that defines the covariance of the Gaussian prior over θ_k. To simplify the presentation, we denote $\mathbb{E}_{\Theta} f_k(y_{i-1}, y_i, X_n)$ as $\mathbb{E}_{\Theta} f_k$. In order to accelerate the training process, the training data is divided into a number of batches. $g(\theta_k)$ is computed by processing one batch in one iteration and switching to another in the next iteration. Each batch consists of N_b sequences, normally $N_b \ll N$. $\sum_{n=1}^{N_b} \sum_{i=1}^{I_n} f_k(y_{i-1}, y_i, X_n)$ is the feature count that f_k is activated in the batch. $\mathbb{E}_{\Theta} f_k$ is the expected feature (f_k) count which is calculated by using CRFs with the parameter Θ at the position i in the sequence n.

$$\mathbb{E}_{\Theta} f_k = p''(y_{i-1}, y_i|X_n, \Theta) f_k(y_{i-1}, y_i, X_n) \tag{5.11}$$

In Equation 5.11, $p''(y_{i-1}, y_i|X_n, \Theta)$ can be calculated by a forward and backward method mentioned in [45]. The idea is to inference in CRFs using Equation 5.1 to calculate the accumulated distribution of all possible paths that pass $\{y_{i-1}, y_i\}$.

The forward vector is initialized to $\alpha_0 = V_s$. It can be inferenced in below.

[1] An open source CRF toolkit at http://crfpp.sourceforge.net/

$$\alpha_i(X) = \alpha_{i-1}(X)M_i(X) \tag{5.12}$$

The backward vector is initialized to $\beta_{I+1} = V_e$, and can be inferenced by the following equation.

$$\beta_i(X) = M_i(X)\beta_{i+1}(X) \tag{5.13}$$

With the definitions of the forward and backward vectors,

$$p''(y_{i-1}, y_i|X_n, \Theta) = \frac{\alpha_{i-2}m_{i,y_{i-1},y_i}\beta_i}{Z_\Theta(X_n)} \tag{5.14}$$

The parameter vector Θ can be iteratively updated by moving a small step s along the directions given by the gains which can be calculated via Equation 5.10. $Z_\Theta(X_n)$ is calculated via Equation 5.5 by appling Θ to CRFs. Viswanathan et al. developed a Stochastic Meta Descent (SMD) method [77] in which the step value s is updated iteratively as a diagonal conditioner. In addition to the gain (first order derivative), this method requires computing the Hessian (second order derivative) of the target distribution. A Hessian vector product is computed to avoid the difficulties of directly computing Hessian. We repeat the method in the following equations where t is the iteration counter.

$$\theta_{k,t+1} = \theta_{k,t} - s_{k,t}g(\theta_{k,t}) \tag{5.15}$$

$$s_{k,t+1} = s_{k,t} - \max(\frac{1}{2}, 1 - \mu g(\theta_{k,t+1})v_{k,t+1}) \tag{5.16}$$

$$v_{k,t+1} = \lambda v_{k,t} - s_{k,t}(g(\theta_{k,t+1}) + \lambda H_{k,t}v_{k,t}) \tag{5.17}$$

In Equation 5.16 and 5.17, λ and μ are two scalars chosen according to the training problem. The Hessian vector product $H_{k,t}v_{k,t}$ can be approximated as follows.

$$H_{k,t}v_{k,t} = \frac{g(\theta_{k,t} + \epsilon v_{k,t}) - g(\theta_{k,t})}{\epsilon} \tag{5.18}$$

For the detail about the SMD algorithm, please refer to [77].

5.2.3 Feature Induction of CRFs

Feature induction of CRFs was first introduced by Mc- Callum in 2003 [53]. As training CRFs requires considerable computational power, the induction is mainly about how to define some more efficient evaluations for incrementally inducing the feature functions of CRFs. The method of McCallum was experimented on name entity recognition and noun phrases segmentation. It resulted in comparable prediction accuracy to the approaches other than CRFs [53]. There are a few further works on the topic. In 2006, Dietterich et al. embodied the boosting algorithm for simultaneously inducing features and training CRFs [27]. In 2007, Liao et el. developed the boosting approach further by integrating the belief propagation [48]. All these works experimented on the same synthetic data which served as a testbed for the comparison.

Feature induction is a difficult problem because training CRFs requires considerable computational power. But it is the most accurate way to evaluate the induced feature set. The number of candidates that can be included in the feature functions grows exponentially by several degrees: the number of atomic elements in the observations, the number of conjunctions in a feature function, and the number of feature functions in the CRF model. Therefore, in order to develop an effective feature induction algorithm, one or more layers of the efficient evaluations need to be established between the raw observations X and the final CRFs.

In McCallum's approach, there are mainly three layers of the evaluations [53] shown as three rectangles in the center of Figure 5.10. From top to bottom, each step to the lower layer requires more computational power of several levels of significance. Therefore, the amount of the candidates that each layer can process decreases significantly.

In *observation test*, the misclassified nodes are used to compute the candidates. This is achieved by simply counting: the more frequent a feature appears in the misclassified nodes, the better the chance that the feature can be added to the candidates. In *candidate evaluation*, each candidate is evaluated by measuring how much it can increase the gain $G(f_{K+1})$, which can be calculated by the following equation.

$$G(f_{K+1}) = \max_{\theta_{K+1}} \sum_{n=1}^{N} (p_{f_{K+1}}(Y_n|X_n) - p(Y_n|X_n)) \tag{5.19}$$

In Equation 5.19, p is from the current CRFs with feature functions $\{f_1, f_2, ..., f_K\}$ and parameters $\Theta = \{\theta_1, \theta_2, ..., \theta_K\}$. Based on p, $p_{f_{K+1}}$ is from the CRFs that includes an extra candidate f_{K+1}. Its weight θ_{K+1} can

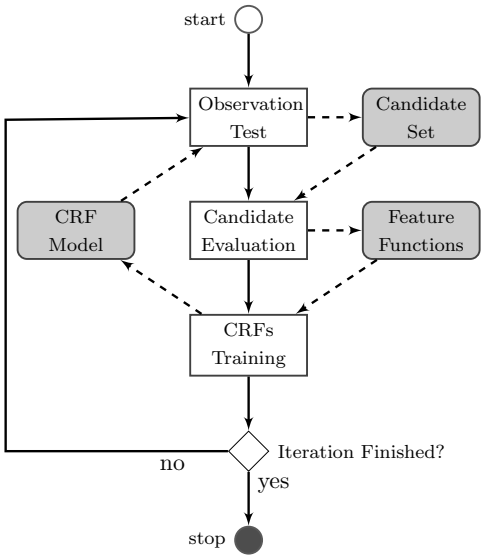


Fig. 5.10. The Feature Induction of CRFs

be calculated by fixing Θ so that the evaluation can be done much faster than training the whole CRFs. Normally, θ_{K+1} can be calculated by a few iterations of quasi-Newton method.

A few candidates with the highest evaluations can be merged with the original set of feature functions. In *CRFs training*, all the parameters are updated for the new feature function set. The resulted CRFs is used to compute the misclassified nodes which can start another iteration at *observation test*.

In summary, the definition, the training, and the feature induction of the CRFs are addressed as the preliminaries in this section. We will explain our own feature induction method in Chapter 7.

6

Observing Human Behaviors in Table Soccer

Playing table soccer requires fast reaction, accurate movements, and skillful actions. Starting from the initial situation with a stationary ball, professional players can perform the complex action sequence "slide-kick", which includes a "pass" and a "kick" to attack the goal, within 0.3 seconds, ending with the ball in the goal. The ball can easily reach a speed of $10m/s$ in the game.

Our robot KiRo can play one side of a table soccer game autonomously. Cooperating with a company, we developed KiRo further [56]. A commercial product called StarKick [78] was developed in 2005 which can win 90% of all the games against average human players. However, StarKick is completely reactive lacking any level of sophistication.

In order to act against human actions, as well as to attack the weakness of human players, we designed mechanisms for recording and segmenting games played by human players.

In Section 6.1, the construction of a table soccer game recorder is sketched. A practical and easy approach is implemented to obtain accurate measurements with high frequency from a game table. Based on that, we describe a finite state machine which we use to segment the recorded data. The approach that explores the properties of the recorded data can be implemented. We show in a small experiment that this approach works reasonably, but it has some limitations.

In Section 6.2, the issues of tracking the game ball are addressed. Several models and learners are developed to work together in the SAL framework. In the experiments, the tracking tasks were successfully achieved and the results are satisfying.

6.1 A Table Soccer Game Recorder: KiRe

Recording table-soccer games requires measurements of the angle and position of the game rods and estimations of the position of the ball. An overhead camera is used in KiRo to observe the ball with a frame rate of $50Hz$ [79]. Eight DC motors are used to control four game rods on one side in KiRo. The positions and the angles of the computer controlled rods are provided by the motor encoders. The angles and positions of the rods controlled by the human player are only very crudely estimated using the camera. The existing system can be used for recording, however, the accuracy of the sensors, the speed of the movements, and the required processing power are challenging. And ball detection is hard if it is hidden from the camera. For example, if we want to use the overhead camera to record the "slide-kick" action sequence which is mentioned in the beginning, there are only 6 frames recorded for this action sequence. It is very hard to either imitate or explain the action sequence based on these recorded frames [89].

6.1.1 Observing the Game Ball

We use a laser range finder to estimate the position of the ball. As it is very expensive to build sensors and mechanical parts from scratch, our recorder is mainly made from parts available on the market. Thus, our game recorder balances cost and quality.

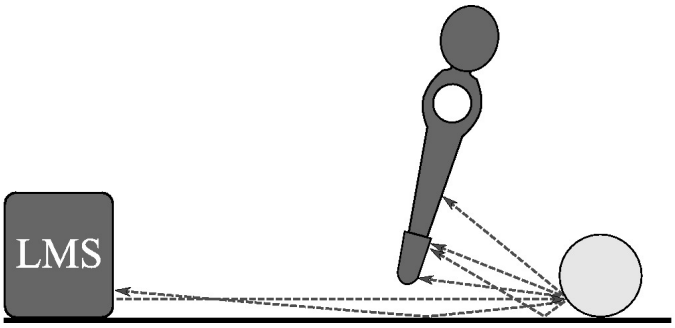


Fig. 6.1. Laser Measurement System

As the basis of our development, a game table was purchased. We have chosen the SICK $LMS400$ laser scanner for observing the ball. It has a frame rate of $350Hz$ and an angular resolution of 0.25 degrees which means that the

data is much better than that of an ordinary camera. As there is a gap of $12mm$ between the playing surface and the end of a playing figure, and the valid range of the sensor is from $700mm$ to $3000mm$, we set $LMS400$ behind the goal of the game table so that the laser beam can go through the gap, shot on the lower part of the ball without any disturbance from the playing figures. Figure 6.1 illustrates how the laser measurement system works.

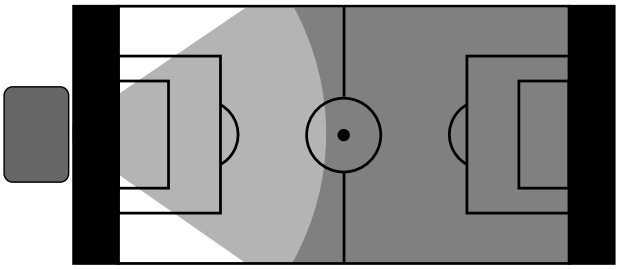


Fig. 6.2. Measuring by One LMS

As mentioned before, one LMS cannot cover the whole playing surface because of its valid measurement range as shown in Figure 6.2. In order to cover the whole surface, we set up the second LMS as illustrated in figure 6.3. The two LMSs can cover the whole playing surface. And they are synchronized to avoid interference with each other.

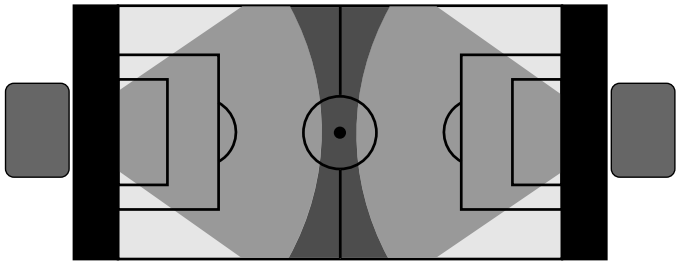
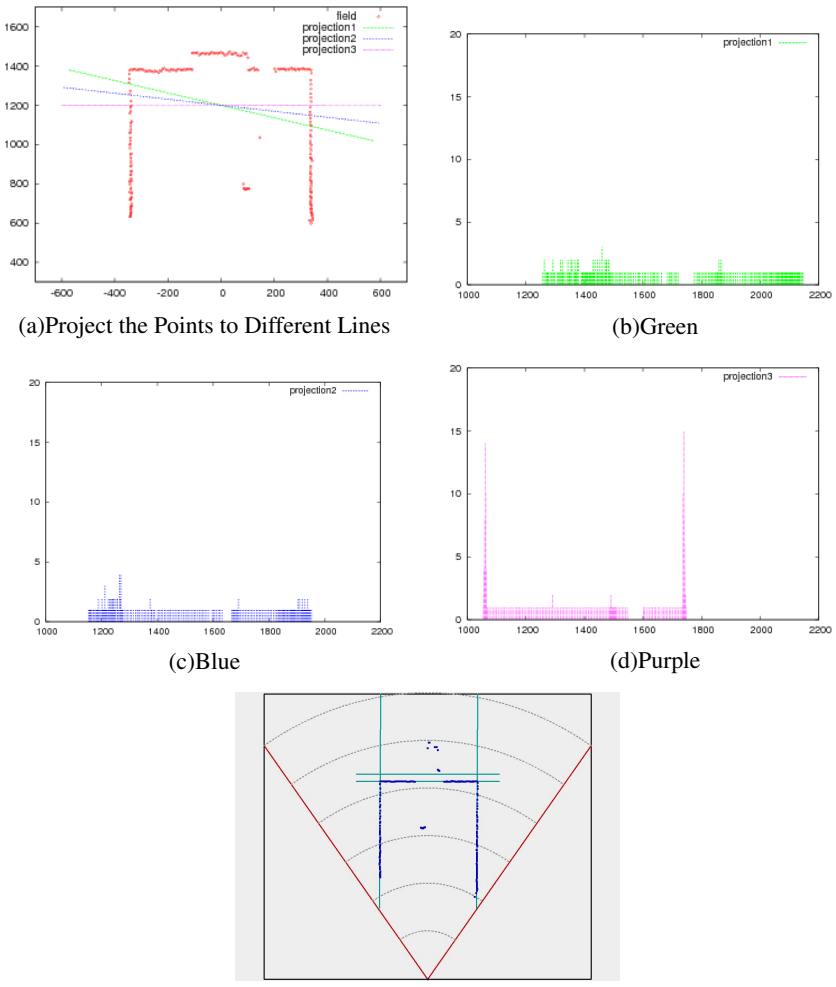


Fig. 6.3. Measuring by Two LMSs

We implemented a calibration program which created a two dimensional coordinate system linked to the playing surface. Consequently, the raw sensor data can be transferred to the coordinates. As explained before, the laser beam of $LMS400$ is reflected by the lower part of the ball. The walls around the playing field form the "background" for the laser beam. The information



(a)Project the Points to Different Lines (b)Green

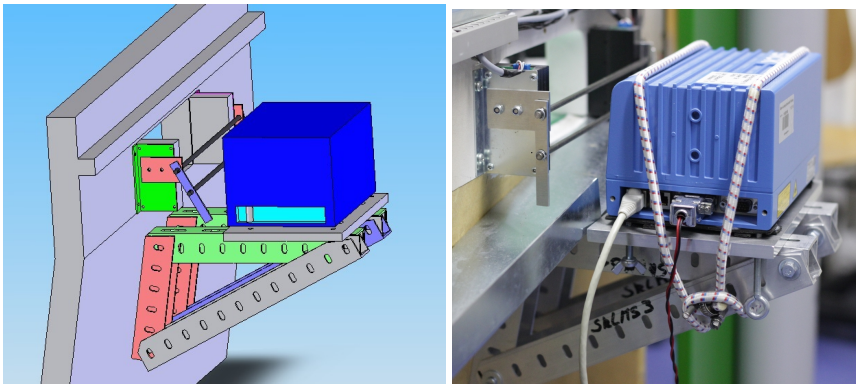
(c)Blue (d)Purple

(e) The Calibrated Field

Fig. 6.4. The Calibration of the LMS on the Game Field

about the ball position can be estimated by removing the background from the raw sensor data. And the coordinates can be calculated by measuring the distance between the ball and the boundaries. The calibration process for the $LMS400$ detects the boundaries and the background which are the border lines matching the data points of $LMS400$. An algorithm was implemented to project all data points onto a line. If the line is perpendicular to the target boundaries, there should be a peak of projected data points for each boundary. The calibration algorithm thus searches for a line which leads to the highest

peak of the projected data points. Figure 6.4 shows the three “horizontal” ines and the prejections on them.



The Tray of LMS (Design) The Tray of LMS

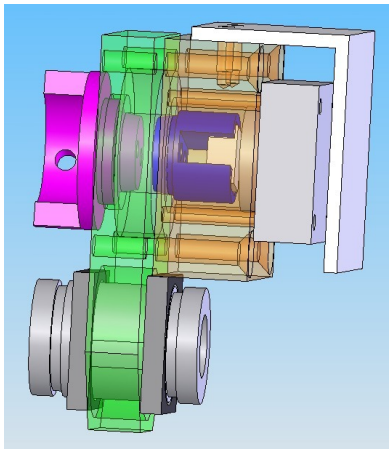
Fig. 6.5. The Designs and the Real Frame for Supporting LMS

The mechanical frame supporting LMS were developed by using SolideWorks – a Computer Aided Design and Manuafacture (CAD/CAM) software. Figure 6.5 shows the designed frame and the implemented frame. We created a net in front of the LMS to protect the opening of the laser from being broken by the game ball.

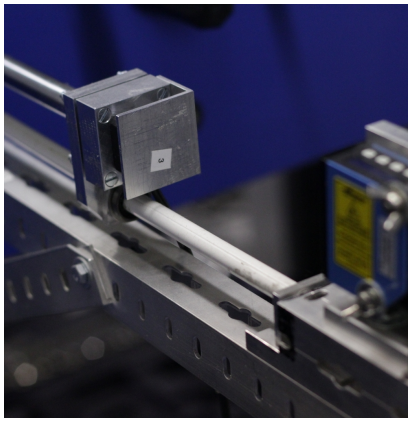
6.1.2 Observing the Game Rods

A rotary encoder and a distance sensor are to estimate the movement of each rod. The linear position of a game rod is estimated by a SICK $DT20$. It is an optical distance sensor which has an accuracy of $1.5mm$ and it can measure the distance without any physical contact to the target. A rotary encoder is used to observe the turning of the rod. It is a magnetic turning encoder which can measure the orientation of the magnetic stick without touching it. A slide-joint unit is installed between the game rod and the rotary encoder. Thus the game rod can be moved as usual along the length direction while the turning is transferred to the rotary encoder

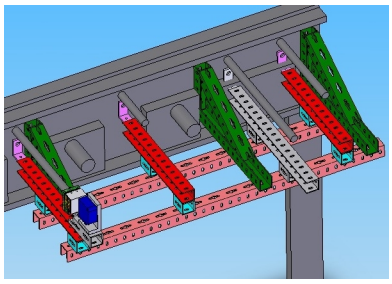
The mechanical units supporting the sensors were also developed using SolideWorks. There are mainly three challenges: the rods can be moved and turned at the same time; the measuring system should not affect the feeling of playing games; the space for the measuring systems is very limited. Figure 6.6 shows the implemented recording system. The left column of the figures



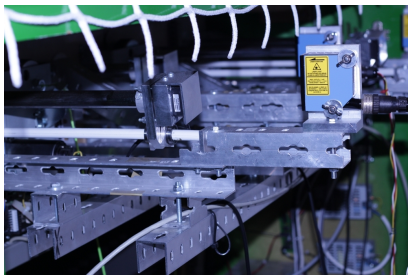
The Turning Encoder (Design)



The Turning Encoder



The Frame of Star-Kick (Design)



The Frame of Star-Kick

Fig. 6.6. The Designs and the Real Measurement Systems

are the designs in the software and the right column shows the picutures of the real system.

Calibrating the distance sensor is simple. The maximum and minimum data points were recorded by moving the game rod to each end manually. The position of the rod was calculated by a linear function. Similar to the calibration of the distance sensor, the game rod was set upright manually. This position was recorded as zero in the sensor data. The rod angle was calculated relative to this position later.

We installed the distance sensors and rotary encoders on the attackers defenders and goalkeepers. The accuracy of the data was checked by putting the ball at several specific points on the playing surface and observing the relative position of the ball and the game rods. The specific points can be the center point or the corners of the field lines. The sensor data was visualized on a screen to match the physical situation. A sequence of actions including

several dribble and slide-kick actions was recorded and saved as a log file. The sequence lasted for 46.62 seconds. There were 13004 ball position estimations, 13644 rod positions and 9816 rod angle estimations. Thus the frame rate was more than $210Hz$.

6.1.3 Segmenting the Recorded Games

An intuitive approach using finite state machines was implemented to segment the recorded data. A small experiment was done in order to investigate the properties of the recorded data and set a base line for further experiments.

When a human is playing table soccer, the intentions are typically "stopping", "dribbling", and "attacking". "Stopping" means getting control of the ball. "Dribbling" is a preparation for "attacking" during which the ball is passed to a comfortable position which is always the start point of an "attacking". "Attacking" contains a sequence of action, such as a "slide-kick", ending at kicking the ball towards the goal. In this dissertation, the tasks of segmentation is defined as segmenting the recorded sensor data according to the three intentions defined above.

There are several challenges in segmentation. Firstly, we need a bridge between the inputs and the outputs. The input of the segmenting system is a sequence of coordinates and angles with time-stamps. The output labels, "stopping", "dribbling", and "attacking" are abstract symbolic data. Secondly, the segmentation should balance efficiency and quality. The sensors provide high-frequency data with noises. The segmentation should work in an on-line manner, being tolerant to noise. Creating a data set which is sufficient for training the target model, is another challenge. Training a segmentation model normally requires a training set to help a machine learning algorithm to adjust the parameters of a model automatically.

Many machine learning algorithms are based on features with discrete values. Our basic idea is to transform the recorded coordinates and angles to a set of discrete features. A figure who is controlling the ball or is going to control the ball is defined as "active". The spatial relations between the game ball and the active figure are the most important feature in our implementation. 13 relations are defined here. They are left-align, right-align, forward-align, backward-align, left-touch, right-touch, forward-touch, backward-touch, positive-lock, negative-lock, left-detach, right-detach, and irrelevant. Figure 6.7 shows the first eight of these relationships. If the ball and the active figure are not aligned in any direction, the situation is considered as "detach". If the ball is on the left of the active figure, it is Left-detach.

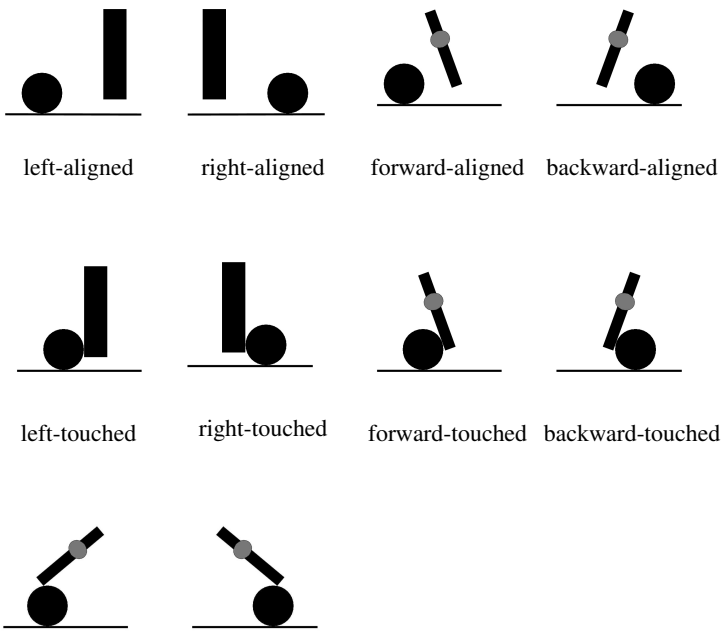


Fig. 6.7. The Spatial Relations of the Ball and the Figure

Otherwise it is right-detach. Irrelevant means the ball is not within the control range of the active figure.

So far, an action sequence can be transformed into a sequence of states. We merge the successive data-slices together if they have the same spatial-relation value. For example a "slide-kick" action could start at positive-locked, through pass left-detached, left-aligned, left-detached, forward-aligned, forward-touched, and finally end at forward-aligned. Each relation mentioned here could include several data-slices. And extra features such as start and end time-points are added to create a "state". Based on the states, a set of finite state automatons were constructed. Each of them can recognize one of the following primitive actions: "active-rod changed", "stop", "touch", and "kick". Active-rod change is the situation where the ball moves from one game rod to another. Stop means the ball is either locked or still for a short time period. Touch and kick are the actions where the ball is either side-touched or kicked by the active figure. We added some limitations on the duration of a

primitive action which reduced the noise and got rid of actions that were very slow. Based on the ability to recognize primitive actions, the segmentation was finally implemented according to the following rules. Stopping starts at an active-rod changed and ends at a stop. Dribbling contains two or more succeeded stop. Attacking starts at a stop, pass a kick, and end at an active rod change.

The experiments were based on the mentioned log file. We run the algorithm to segment the log data into target action sequences. The segmented data were replayed and checked manually. No any sequence was missed and no any unwanted sequence was added in the result. The computer used for this caculation has a $3.2GHz$ CPU and $1G$ memory. The log file was played at its real speed. The algorithm finished the tasks as soon as the log playing stopped. The CPU monitor did not show any processing pressure during the process.

6.1.4 The Software Architecture

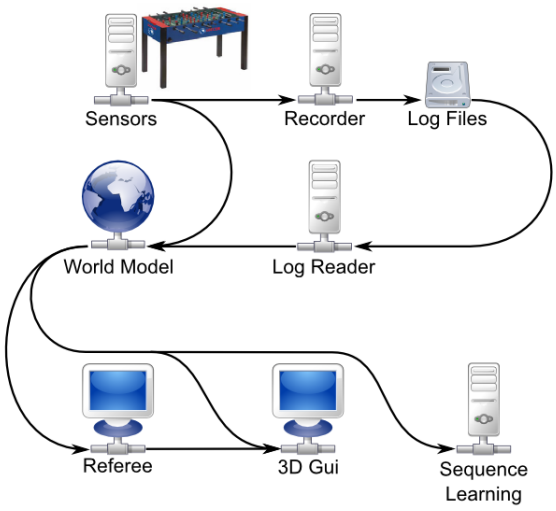


Fig. 6.8. The System Components

There are mainly eight components in KiRe. Figure 6.8 shows the system architecture. The components can be connected with each other via Ethernet. They can either run as independent processes in one computer or be dis-

tributed in several computers. This architecture has the advantage to isolate the problems. For example, displaying and world model are two components that can be connected with each other. In case "world model" has a problem, we can simply run the minimum components to debug the problem. In addition, the further developments can be easily integrated into the system by adding more components.

The components and their functionalitis are given in the following list.

- Sensors: The program which can read and write the real sensors.
- Recorder: The program that saves the sensor data into hard disk.
- Log files: the recorded data.
- World model: the table soccer "world" observed by the computer.
- Log reader: the reader that can read and replay the log files.
- Referee: an application to detect the rule violation automatically.
- 3D GUI: the graphic user interface showing the table soccer game in 3D.
- Sequence learning: the learning algorithms for table soccer.

6.1.5 Discussion

Our segmenting algorithm uses a layered structure to build a bridge between the low-level sensor data to high-level action sequence labels. The noise is filtered out by using domain knowledge which is mainly expressed as the spatial relations and the duration of the actions. Although the segmentation works fine with the log file, it still has some limitations. The algorithm which maps continuous coordinates and angles to spatial relations has "value oscillating" problems near boundaries. Many parameters were defined manually which made the algorithm hard to maintain. It is not easy to extend the approach for other interesting applications such as an on-line prediction.

The slolution would be to remove the thresholds and boundaries using probability models. Hidden Markov Models (HMMs) could be one of the solutions. HMMs are based on Markov chains and statistics [7]. They were first successfully used in speech recognition [5] and became wide spread in many domains. For example, HMMs can be used to model DNA sequences in bio-informatics [44]. They are employed to describe the return series in finance [60] and they can also help to navigate a robot [67].

As the existing learning algorithms for HMMs search for a local maximum of the likelihood, it would be interesting to try and compare the following two approaches using HMMs in the context of our research. On one hand, the approach using a fully automated method [33] could be implemented. On the other hand, HMMs are constructed by using known information as much

as possible. In this approach, the states of HMMs can be defined using the described spatial relations. Based on the knowledge of the existing transition models [58], we could choose a left-right model in our implementation. The observations are with continuous values. The probability distributions of the HMMs can be initialized by some supervised learning approaches. The states could be refined using Viterbi algorithm. And finally the Baum-Welch algorithm can be used to enforce convergence of the parameters.

6.2 Tracking the Game Ball in Table Soccer Using Switching Attention Learning

KiRe is a table soccer game recorder. It provides a way to record human-played games [88]. In KiRe, the position of the ball is measured by a laser measurement system (SICK $LMS400$) as illustrated in Figure 6.1. We installed the LMS behind the goal of the game table. The laser beam goes through the gap and targets at the lower part of the ball. The receiver at the upper part of the opening on the LMS receives the reflected signals. The opening angle of the laser beam is 70°, and the valid measurement range of the LMS is from $700mm$ to $3000mm$. Each data slice contains 280 measurements of distance and angle evenly distributed over the open angle. Figure 6.2 shows the bird-view of the game field. With one LMS, the field is divided into three different regions. Two corners in the left side are outside the view of the laser. The measurements are invalid in the grey fan region. The remaining dark grey region is within the valid range of the LMS, covering the right half of the field. By removing the background from the laser view, we can compute the position of the ball.

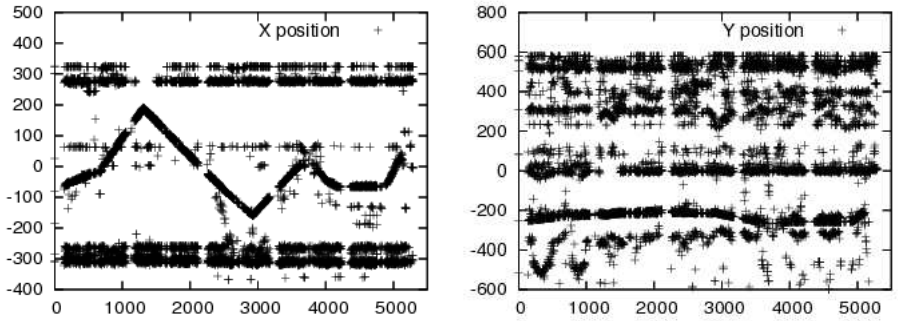
In order to measure the whole field, we mounted two LMSs symmetrically in KiRe. These two LMSs communicate via Ethernet, which synchronizes their laser scans. Figure 6.3 shows the different regions in this situation. In Table 6.1, we list the types of the regions and the information for locating the ball. When the ball is in the grey regions, one of the LMSs gets the invalid distances at the laser spots on the ball. However, the angle of these invalid data still provide information about the position of the ball. Therefore, we can still fuse the measurements from two LMSs to compute the intended position.

The LMSs scan the field with a frequency of $350Hz$. With two LMSs synchronized, the computer needs to process 700 data frames per second. We recorded a segment of data in about 5 seconds during which a human player dribbles and kicks the ball. Figure 6.9 shows the recorded data. The horizontal axis shows the time in million seconds. The vertical axis shows the position

Color	Description
Dark Grey	Both LMSs have valid measurements
Grey	One LMS is valid, the other is invalid
White	One LMS is valid, the region is out of the other's view.

Table 6.1. Regions and Their Information

in million meters. The x and y positions are shown in separate plots. There are several possible positions in a single data slice because of the noise. Thus we got more than 7000 points all together.



(a) X Positions (b) Y Positions

Fig. 6.9. Raw Data from the Sensors

The raw data is very noisy which makes the tracking task very challenging. Below we list the main difficulties.

- When two LMSs are face to face, they interfere with each other even if they are synchronized. The laser beam of one LMS is reflected by the mirror of the other.
- The game figures, which are moved and turned constantly, disturb the reception of the laser signals.
- When the ball jumps, it cannot be observed by the LMSs
- We do not use a real-time operating system. There are some processing gaps of about $80ms$ in the data.

We developed a **ball module** using SAL for the tracking task as shown in Figure 6.10. There are three models in the module. A **sensor model** is constructed for filtering the noise. A point in a data slice is classified as valid or

noise. The noise will be discarded. Each data slice contains one valid point at most. A **segment model** is developed for segmenting the data sequence into much smaller parts. In each segment, the ball is assumed to have constant acceleration and movement direction. A set of **Kalman filters** are implemented to smooth the data within a segment. The raw data is processed first by the *sensor model*. The sequence of the valid points is forwarded to the *segment model*. When a new segment is detected, the *Kalman filters* are *reset* so that they can adjust to the sudden changes in the movement of the ball.

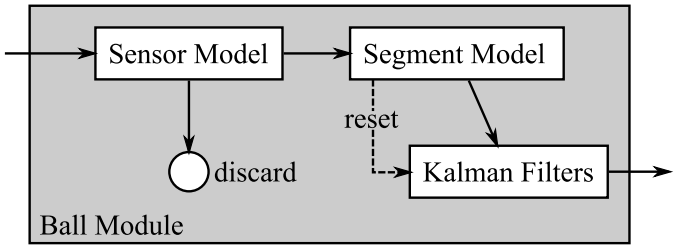


Fig. 6.10. The Ball Model

6.2.1 The Sensor Model

An approach based on decision tree is implemented in the sensor model for the classification. Clusters are constructed according to the dynamic updates of the data sequence. In each cluster, the data points are close to each other. With the help of the clusters, the decision tree can be constructed based on Boolean attributes which can be obtained by answering the listed questions.

- Is the data point supported by both LMSs? The point is called a **full-belief point** in the true case.
- Does the data point belong to a cluster which has the maximum point number among all the clusters?
- Does the data point belong to a cluster which contains full-belief points?
- Is the point in a region where the noise points are detected with high probability? The four corners are this kind of region because each corner can be observed by only one LMS.
- Does the point belong to a cluster which is updated very often?
- Does the point belong to a newly-created cluster?
- Will the point generate high innovation if it is used to update the Kalman filters?

6.2.2 Kalman Filters and Segmentation

The data from LMSs is a distance-angle pair. We need to transform them to (x, y) position. Because of the transformation, extended KFs should be employed for the tracking. However, we choose discrete KFs. Discrete KFs save computational power significantly, when they are used separately for x and y. And they do not perform worse than extended KFs in the experiments. The discrete KFs are updated according to the time and measurement update functions which are already standard [84]. We skip them here.

By considering the position, velocity and acceleration, we implemented an approach using *triple-integrator* KFs which is widely used and has excellent performance in many applications [8]. Here we only explain it briefly. Equation 6.1 defines the vector of x, where v_x is the velocity along x direction, a_x is the acceleration. The update of $\mathbf{X}$ is governed by Equation 6.2, where A_x is defined in Equation 6.3. Δt is the time span since last data slice. The process noise w_k is defined in Equation 6.4, where Q matrix can be computed dynamically by Equation 6.5. The belief factor b can be adjusted to trade-off the prediction and the measurement.

$$\mathbf{X} = \begin{pmatrix} x & v_x & a_x \end{pmatrix}^T \tag{6.1}$$

$$\mathbf{X}_k = A_x \mathbf{X}_{k-1} + w_{k-1} \tag{6.2}$$

$$A_x = \begin{pmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{pmatrix} \tag{6.3}$$

$$p(w) \sim N(0, Q) \tag{6.4}$$

$$Q = b \cdot \begin{pmatrix} \frac{1}{20}\Delta t^5 & \frac{1}{8}\Delta t^4 & \frac{1}{6}\Delta t^3 \\ \frac{1}{8}\Delta t^4 & \frac{1}{3}\Delta t^3 & \frac{1}{2}\Delta t^2 \\ \frac{1}{6}\Delta t^3 & \frac{1}{2}\Delta t^2 & \Delta t \end{pmatrix} \tag{6.5}$$

6.2.3 The Evaluation Model

The ball module has the functionality of classifying, segmenting, and smoothing the data sequence. These functionalities are based on the data from the past. From another point of view, we can evaluate the performance of the ball module without any temporal limitation. For example, both the data from the past and the data from the future can be used for classifying the current point. According to this principle, we implement an evaluation model which follows three rules listed below.

- A valid data segment should contain at least three full-belief points.
- A point belongs to a data segment if its distance to the closest point in the segment is within $3mm$.
- The more points are included, the better the performance the system gets.

6.2.4 The Active Elements and the Learners

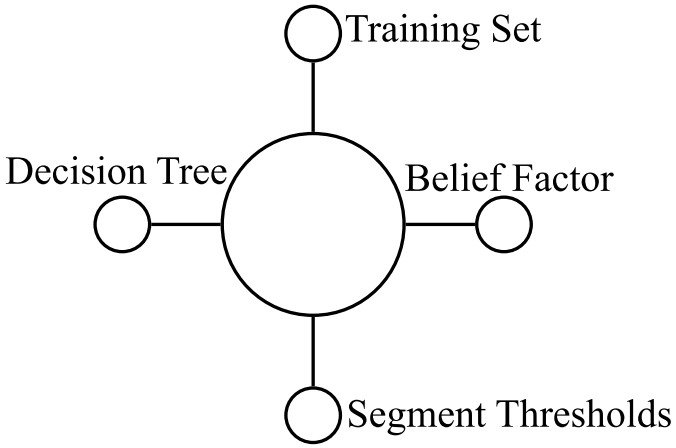


Fig. 6.11. The Active Elements

With the help of the evaluation model, we can develop several learners to improve the performance of the system. As the medium for the improvement, four *active elements* are found. They are shown in Figure 6.11. **Decision tree** is used for the classification in the sensor model. **Segment thresholds** are time and innovation. **Belief factor** trades off the prediction and measurements in triple-integrator KFs. This factor affects the innovation threshold used in the segment model, which will reset KFs when a data point generates a high-grade innovation. **Training set** can be regarded as the results of the evaluation. Each example in the set contains seven attributes and a class label. The attributes are provided by the sensor model. The class label comes from the evaluation. Four learners are implemented to learn the active elements from the data. The learning tasks are listed as below.

1. Learning the training set from the output of the ball module
2. Learning the decision tree from the training set
3. Learning the segmentation thresholds by maximizing the point number in the output of the ball module.

4. Learning the belief factor by minimizing the average innovation of the KFs

As there are not so many learners, we developed a simple attention model in which each iteration has seven steps: E, 1, 2, E, 3, E, 4. In an **E** step, the performance of the system is evaluated.

6.2.5 The Experiments

We use the recorded data shown in Figure 6.9 for offline learning. The implemented SAL system is run on the data for six iterations. The results are shown in Figure 6.12. The first row is the valid trajectory of the ball which is obtained manually. The second row shows the output of the learned system. Although there are a few noise-points left in the trajectory, the results are satisfying. This noise remains because the decision tree does not have enough attributes to classify them apart from the valid points. The third row shows the improvements over the iterations. In this figure, the horizontal axis shows the number of the *E* steps performed. The solid curve shows the changes of the output number of the ball module. The doted line shows the changes of the valid point number in the evaluation. After about six *E* steps, which are two iterations, the learning converges to the satisfying results. The learning curves indicate that a better ball module enlarges the ability of the evaluation, while a better evaluation improves the ball module.

The trained system is tested in an online manner. We found that the ball could be tracked in real-time and its trajectory was smooth enough. A segment of data is illustrated in the first row of Figure 6.13. The output of the system is shown in the second row. In the figure, the ball is lost when it is in one of the corners, because the corners are regarded as noisy regions which are not important for the game. Therefore we can conclude that the trained system stays stable on the unknown data.

6.2.6 Discussion

The first application of SAL was implemented to track the game ball in table soccer. There are four active elements in the application, a decision tree for classification, the training set for building the decision tree, the belief factors for KFs, and the thresholds for segmentation. We developed four learners for these elements and implemented an attention model for the iterations over these learners. The experiments showed that the application was successful. The intended improvement space was generated by the learners in the system

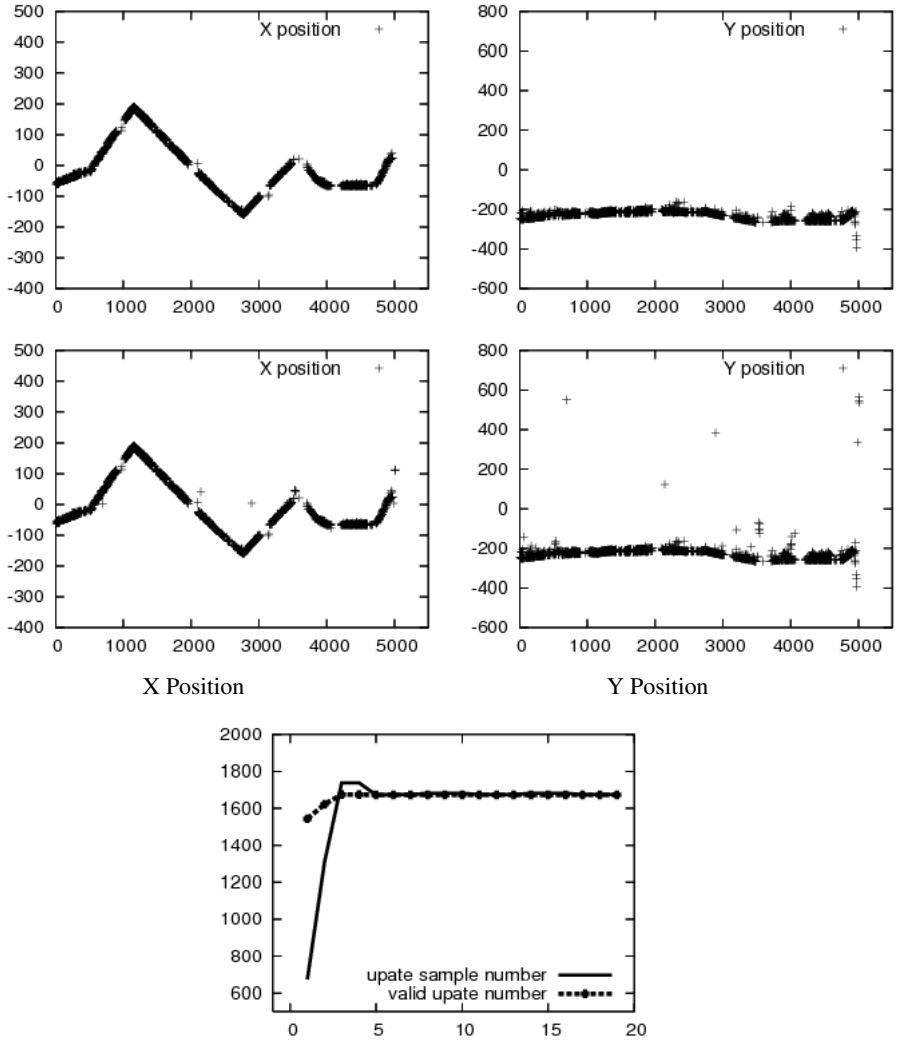


Fig. 6.12. Offline Learning

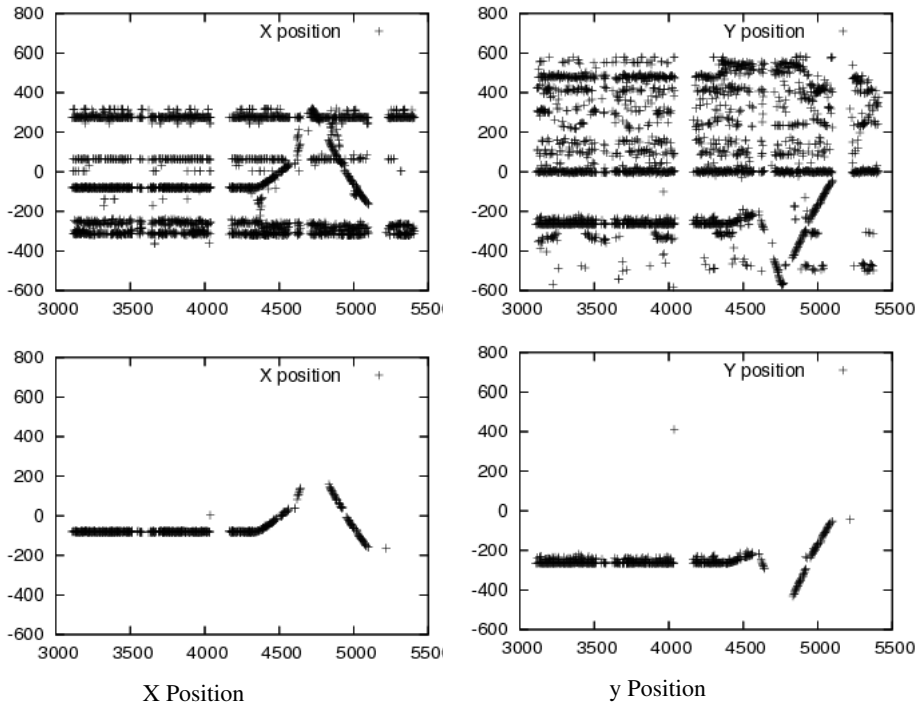


Fig. 6.13. The Selected Data Segment in Online Testing

7

Constructing Conditional Random Fields Using Switching Attention Learning

Our research in CRFs was originally motivated by the explanation of the data of table soccer games. The sequential game data is made available via a game recorder [88]. The labels are the actions of human players e.g. lock, attack, block, pass, and dribble. These data considerablly differ from the synthetic data. At each time slice, the measurements are encoded in a vector of Boolean variables. Each skill of human players consists of hundreds of such time slices. The annotation of the data is very exhausting. We have spent about 80 hours annotating 200 sequences, which are much too little for the feature induction and the supervised learning of CRFs. This difficulty motivated the idea of the simulation.

A sequence generator was built to create data sequences, and to label them automatically, simulating the data of the table soccer games. The core idea is to create the first CRFs, with the feature functions and the parameters generated randomly. Then, the second CRFs can be obtained from the feature induction methods as mentioned above. The first CRFs can thus provide the information to estimate the second one and the induction algorithm. This approach creates an extra phase to explore several basic properties of the feature induction of linear-chain CRFs. Consequently, it fosters several further developments. The highlights of this work are summarized as follows:

- We list our observations to describe the feature induction issues, which are hard to outline without a simulation.
- We integrate a novel reduction step in the induction, which can keep the accuracy of the prediction and decrease the number of feature functions, thus making the learning more efficient.
- We develop a method to train a queue of CRF models from the data. CRF queues guarantees a no worse prediction accuracy than the single CRFs. It

outperformed the single ones on the data sets in all the configurations. To the best of our knowledge, we are the first who propose the idea of CRF queues.

7.1 Related Works

Variable and feature selection is a well-developed research area. Guyon summarized the issues and the main approaches in the area in 2003 [37]. If we put the specific CRFs problem into a more general context, many ideas and methods can be used. For example, feature reduction is widely employed in this area. To our knowledge, it is not yet applied to CRFs.

We found only a few works about feature induction of CRFs. Chen et al. compared a gradient based approach [22] to the McCallum method [53]. Both approaches use the framework shown in Figure 1. In candidate evaluation, the gradient based approach searches for the candidates that make the objective function decrease fastest. Instead of simply counting in the observation test, some researchers integrated the boosting method [27], [48]. The approaches in this direction can simultaneously induce features and train CRFs, which have the more compact model, and therefore being efficient in the computation.

Our implementation is based on a CRF training algorithm - Stochastic Meta Descent (SMD) [77], and the feature induction framework by McCallum. The experiment platform was implemented according to the descriptions in the publications. We did not use any existing source code from the authors or the open source toolkit via Internet. The main reason is that the sequential data in this work are very different from the data in the synthetic benchmarks. In addition, building a platform from scratch creates more chances to find unique and novel ideas.

We survey the feature induction approaches [53], [27], [48], [22]. These methods were estimated in real applications. A simulation was developed to gain a new phase for the estimation. The induced features can therefore be compared with the target features in the simulation. In addition, all these approaches monotonically increase the number of feature functions. The reduction step is novel in CRFs. Finally, instead of learning a single CRFs, a queue of CRF models can be built. We have not yet found any other approach in the area of CRFs similar to ours.

7.2 A Simulation and the Trained CRFs

There are two parts in this section. The first part introduces the construction of the simulation. The second discusses the basic properties of the trained CRFs.

7.2.1 The Simulation

Our research is based on the configuration of the data considerably different from the data in the benchmarks. The synthetic data are a set of sentences. In the context of CRFs, the atomic variables of these data are the vocabulary which consists of thousands of words. In the simulation, a vector of Boolean variables $B = \{b_1, b_2, ..., b_C\}$ is observed at each time slice. A data sequence consists of a number of such observations. There are only C atomic variables, which are far fewer than the number of words in the vocabulary. These atomic variables, nevertheless, appear very frequently (at each time slice) in the sequences. Labeling (by hand) such a data sequence is very time consuming, even if it is done using a software tool in which the data can be annotated by mouse clicks. Therefore, a sequence generator is developed in this work.

A CRF model describes a stochastic process, which reveals the relations among the observations and the hidden labels. In the training process, the success of the CRFs hints that the acquired stochastic process matches the patterns in the data. First the training data are avaiable, then the CRFs is trained from them. The idea of the simulation goes the retrograde way. First a CRFs is generated, then it can be used to compute the hidden labels of any randomly generated observations. The following is the assumption which bridges the simulation and the simulated process.

- **The stochastic processes in the target system can be described as a CRF model.**

The simulation is shown as the upper row of the boxes in Figure 7.1. There are mainly two algorithms. A CRF model is generated by the *model generator*. The model is used in *sequence generator* for computing the most probable explanation of randomly generated X, the data are divided into a test set and a train set. The simulation can be configured mainly by the variables listed below:

- C_x, the number of atomic Boolean variables.
- C_y, the number of labels.
- Θ, the parameters of the CRFs

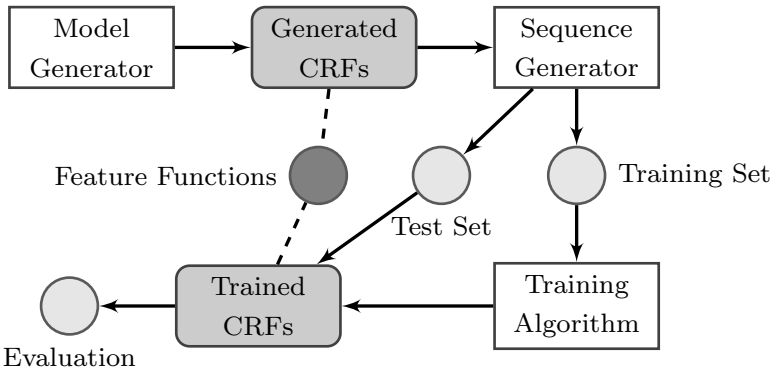


Fig. 7.1. Generated and trained CRFs share the feature functions

- F, the feature functions of the CRFs.
  - C_f, the number of feature functions.
  - C_c, the maximum conjunctions of each feature function.
  - C_o, the maximum offset of the atomic feature.
- C_l, the length of the sequences.

CRFs consists of a set of parameters Θ, and a set of feature functions F. C_x and C_y can be use to configure the difficulties of the problem. For example, the identification of 2 labels by observing the sequences of 200 Boolean variables should not be as difficult as the identification of 200 labels by observing 2 Boolean variables.

In order to prevent a too huge $Z(X)$, $\theta_k \in R$ is drawn evenly from $[0.0, 5.0]$. We do not consider the negative value of a parameter because it causes the corresponding feature function to be hardly observed from the data. In the *observation test*, it would be interesting to consider a feature that can hardly be observed in the sequences. We have implemented the idea, but it turned out to be not very effective because the inducing and training algorithm is based on maximum likelihood.

C_f, C_c, and C_o define the feature functions. All the numbers required by the feature functions are drawn evenly from the defined ranges, e.g. y_{i-1}, y_i in $[0, C_y)$, an atomic observation b_i, where i is in $[0, C_x)$, the value V_i of b_i, and an offset value o_i in $[-C_o, C_o]$.

C_l can affect the accuracy of the model because we consider the rate of the correct sequences as the estimation of the learned CRFs. The longer the sequence is, the more possible that the sequence is mislabeled. A sequence will be regarded as being mislabeled, if a node in the sequence has a wrong

Table 7.1. The Configurations of the Simulation

index	Descriptions	Configurations
S1	single and isolated	$C_c = 1$, each b_i appears maximum once in F
S2	single and conjunct	$C_c = 4$, each (b_i, o_i) appears maximum once in F
S3	Boolean and isolated	$C_c = 1$, each (b_i, V_i) appears maximum once in F
S4	Boolean and conjunct	$C_c = 4$, each (b_i, V_i, o_i) appears maximum once in F
S5	high overlapped	$C_c = 4$, $C_x = 10$, no extra check

label. This estimation is stricter than considering only the accuracy of the nodes. We use it because in some applications the sequences of the labels need to be further classified. **The whole sequence is considered to be wrong if there is a single lable in the sequence is misclassified**.

The simulation provides a platform for studying a wide range of CRFs. After we exploring on different situations, 5 configurations are carefully chosen, which are challenging for the induction issues, being not too hard or too easy. The configurations are designed for the comparisons of the different levels of the conjunctions and the inter-dependencies.

The parameters are set to the following values by default. $C_x = 20$, $C_y = 5$, $C_f = 13$, $C_c = 4$, $C_o = 3$, C_l is evenly drawn from $\{5, 6, 7\}$. In each configuration, a few changes are applied, as listed in table 7.1. In the table, "single" means the atomic feature b_i appears a maximum of once. "Conjunct" means $C_c = 4$. "$S5$" has a high-overlapped F because C_x is decreased to 10 in this configuration, so that 13 feature functions contain 13 to 52 atomic features.

In the experiments, we found that the generated CRFs can be trapped, which hints that the predicted labels exclude some values of Y. In order to avoid this situation, we add one more rule in the sequence generator: the CRFs is not valid if the generated sequences exclude some values of Y.

The simulation was run to create the following data: 100 data sets for each configuration S_i. Each set contains a training set of 1000 sequences, and a test set of 500 sequences.

7.2.2 The Trained CRFs

The experiments are based on 8 computers, each with 8 AMD cores at $2.3GHz$ and $32G$ memory. These CPUs are driven by a grid system, on which 64 tasks can be run in parallel. The experiments described in this paper altogether took about 14 days in the grid system.

In the **first experiment**, we assume F is known. Figure 7.1 shows the scenario. The training algorithm (SMD) was run for maximum 10000 iterations (batches) on each data set, so that the resulted CRFs are well-trained. The trained model and the simulated model are compared in three aspects. The results are shown in Figure 7.2

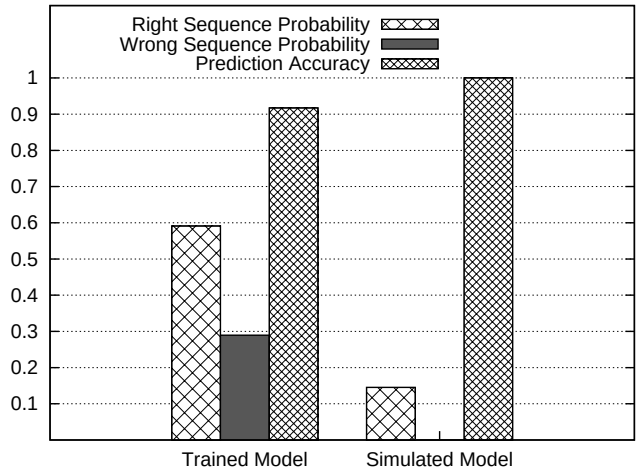


Fig. 7.2. A comparison between the trained model and the simulated Model

The trained models are evaluated on the test sets. The correct sequence probability is the value of $p(Y|X)$, where the Y is a correct label sequence. We denote it as p_r. The wrong probability p_w is defined in the similar way. The prediction accuracy is the rate of the correct sequences. The values in the figure are the average values over all 500 models and the data sets.

The accuracy of the simulated models is 100% because they are used to generate the evaluation data. With the shared feature functions, the average accuracy of the trained models is more than 90%. The trained model has a p_r significantly higher than its p_w and the p_r of the simulated models. We summarize the observations as follows:

- The simulated CRFs cannot be cloned via supervised learning.

- With the shared feature function, the trained CRFs can achieve an average accuracy of about 90%.

We created five different configurations $\{S1, S2, ..., S5\}$ for the simulated model. The detailed information on the trained models over these configurations are shown in Figure 7.3. We inspect the prediction accuracy of the trained models. By using each configuration, 100 simulated models were generated. "Minimum" means the trained model performed worst in the estimation, while "maximum" is the best.

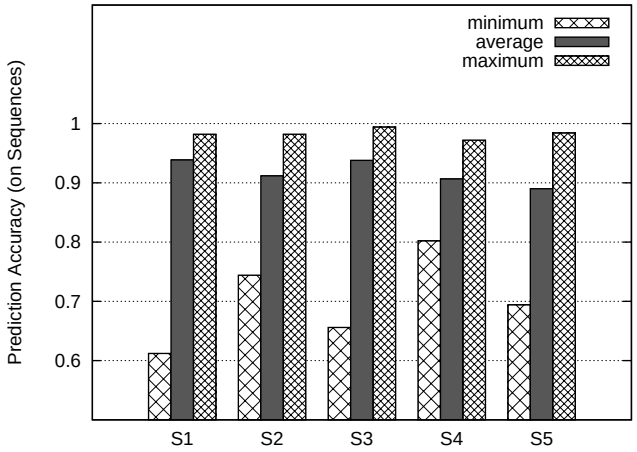


Fig. 7.3. The accuracy of the trained CRFs with shared feature functions on five different configurations

The average performances of the trained models are roughly the same over the five configurations. $\{S2, S4, S5\}$ have the conjunctions in the feature functions. They are up to 5% worse than $\{S1, S3\}$. We could not find a clue to explain the best and the worst performance of the models. For example, the feature functions in $S1$ should not be much different from one model to another because these functions are "single" and "isolated" as shown in Table 7.1. The results can be summarized thus:

- The performance of a trained CRFs depends on both F and Θ of the simulated model. Θ itself has significant impact on the performance.

The trained CRFs can not only compute the most possible explanation for an input X, but also associate this explanation with a probability value. In Figure 7.4, we show the relations between the prediction accuracy and the probability of the most probable paths, which are divided into 10 grids. For

example, the values at 0.15 come from the data sequences with the probabilities of the most possible path from 0.1 to 0.2. The figure consists of two parts. The upper part is about the accuracy of the prediction. The lower part is the distributions of the data in each grid.

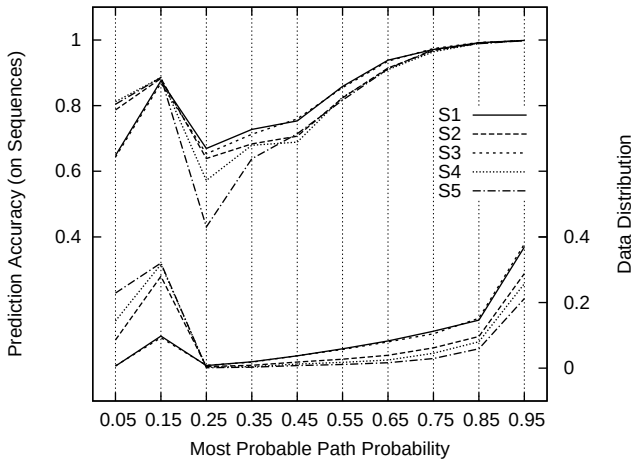


Fig. 7.4. The distributions of the accuracy of the prediction over the probabilities of the most probable path.

These distributions are interesting. Before the training, all parameters of CRFs are initialized to zero. The most possible path of a sequence should have the same probability as other paths of the sequence. If we put the information in the coordinate system in Figure 7.4: all the data should be in the grid from 0.0 to 0.1; the accuracy of the model should be quite low, e.g. near zero. The curves in the figure show the consequences of the training. We can summarized as follows.

- If there is an axis for the probability of the most probable path, the training based on maximum likelihood pushes a large number of sequences in the training set towards the higher value direction of the axis.
- If the axis is divided into two parts at the middle point, the higher part has a higher accuracy than the lower part.

The experiments in this section set a basis for further developments on feature reduction and CRF queue.

7.3 Feature Reduction

As mentioned in Section 5.2.3, the induction process iterates over three steps: observation test, candidate evaluation, and CRF training. In this section, we develop the forth step **feature reduction**.

If the feature functions in the simulated CRFs are considered to be "optimal" for the induction, the observation test and the candidate evaluation can be regarded as the *heuristics* to estimate the values of the possible movements. Apparently, these heuristics are not admissible – thus, from this point of view:

- The behaviors of the induction can be regarded as a local search.

At the "feature reduction", a subset of features F_r is to be removed from F, where F is the set of so far induced features, $F_r \subset F$. For each $f_k \in F$, a gain value G_r is defined as a measurement for the reduction.

$$G_r(f_k) = \sum_{n=1}^{N}(p(Y_n|X_n) - p_{\theta_k=0}(Y_n|X_n)) \tag{7.1}$$

We modified the gain $G(f_{K+1})$ in Equation 5.19 for the reduction. $p_{\theta_k=0}$ means θ_k is set to zero while other parameters are fixed. G_r has the similar meaning to G. The difference is $f_{K+1} \notin F$, while $f_k \in F$. G is calculated in iterations before the traing of the CRFs; G_r can be calculated without any iteration after the training. In the reduction, the features with a G_r lower than a predefined threshold C_0 can be removed.

$$F' = F - \{f_k|G_r(f_k) < C_0\} \tag{7.2}$$

The induction algorithm with feature reduction is written in pseudo-code in Table 7.2. The reduction is called after several iterations of observation test, candidate evaluation, and CRFs training. The algorithm stops after some iterations of the reduction steps.

In the training process, each feature function has a weight. Intuitively, the feature reduction can reduce the number of parameters. Consequently, it should save the computational power required by the training. The **second experiment** is designed for this comparison. The feature induction algorithms are run independently with and without the reduction for 50 iterations. Figure 7.5 shows the results.

In the figure, the horizontal is the function calls of the training. The vertical is average time spent on processing 100 batches of SMD. During the

Table 7.2. Algorithm: Inducing Features with Reduction

	input: Training Examples $(\mathbf{X}, \mathbf{Y})$
	output: CRFs: (F, Θ)
1	$F_{0,I_0} = \emptyset$
2	**for** $i_0 = 1...I_0$ **do**
3	$F_{i_0,0} = F_{i_0-1,I_0}$
4	**for** $i_1 = 1...I_1$ **do**
5	$F_{i_0,i_1} = F_{i_0,i_1-1} \cup$
	{new Features from the *Observation Test*}
6	Compute Θ_{i_0,i_1} on F_{i_0,i_1} via Equation 5.9
7	**end**
8	***Reducing*** F_{i_0,I_1} ***via Equation 7.2***
9	**end**
10	Choose $F = F_{i_0,i_1}$ where F_{i_0,i_1} yield to the best
	performance on the $(\mathbf{X}, \mathbf{Y})$
11	Compute Θ on $\{F_{i_0,i_1}, \Theta_{i_0,i_1}\}$ via Equation 5.9
12	return (F,Θ)

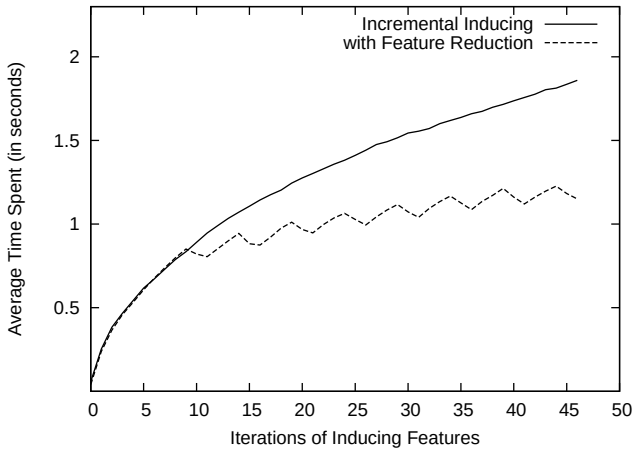


Fig. 7.5. Time spent on 100 training iterations of the CRF models with feature reduction and without feature reduction

induction process, features are added to F incrementally. The training thus requires more and more time to compute the weights of the feature functions. The dotted curve shows the performance of the algorithm with feature reduc-

tion. It is serrated because the reduction step is not called in every iteration. The experiments can be summarized as follows.

- The reduction step can save the computational power more than 30% in the long run.

Although the reduction makes the induction process faster, does it decrease the prediction accuracy of the resulted CRFs? The **third experiment** is designed to investigate this issue. The induction algorithms with and without the reduction were run independently over the 5×100 training sets. The models which yield to the best performance on the training set are selected for the evaluation. The results are illustrated in Figure 7.6.

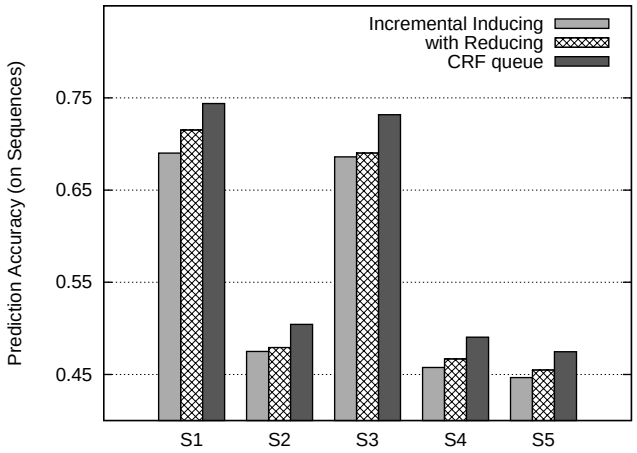


Fig. 7.6. The Evaluation of the Three Approaches over the Five Configurations

In the figure, the results are grouped by the 5 configurations. The evaluation is the prediction accuracy over the sequences, which are averaged over the 100 training sets. For convenience, the performance of the CRF queues is also included, which can be ignored for the moment.

Based on the average results, the algorithm with the reduction outperformed the one without reduction in all the configurations. If we take a closer look, the reduction did not make the algorithm better in any case. The "local search" we mentioned before can explain these results. Either of the two approaches can have better prediction accuracy than the other.

In Figure 7.6, the performances of the models can be roughly classified into two categories: $\{S1, S3\}$ the configurations with single features, and

$\{S2, S4, S5\}$ the configurations with conjunctive features. If we further compare the performances on the configurations with the Boolean and isolated variables, several points can be noted.

- The conjunctive features of the simulated models make the induction tasks more difficult.
- The feature overlapping of the simulated models [only] slightly affects the difficulties of the induction in the experiments.

As illustrated in Figure 7.5, the algorithm with the reduction runs faster because the number of feature functions is lower. How many features were induced in the experiments? In the upper part of Figure 7.7, we show the results.

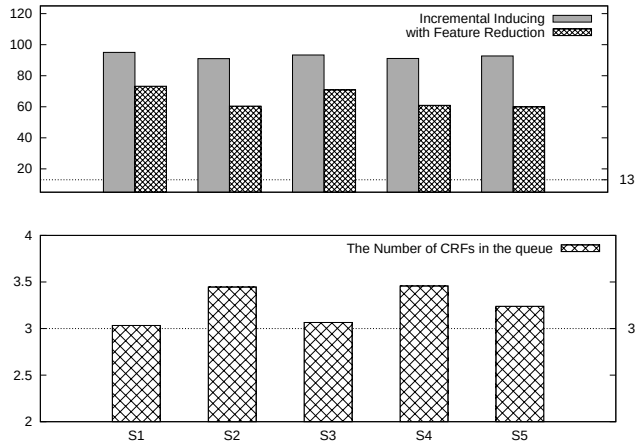


Fig. 7.7. The number of the feature functions and the number of the CRFs in the queue

The data were grouped by the five configurations, in each group the numbers of feature functions are averaged. The "13" comes from the simulated models, which serves as a based line for the comparison.

- The learning induced the features several times more than the target features in the simulated models. Surprisingly, it did not cause a severe overfitting problem.
- For the incremental induction, the average numbers of induced features were not very different from one configuration to another.
- For the induction with the reduction, compared to the configurations with conjunctive features, more features were induced in the configurations with a single feature.

7.4 CRF Queue

In Section 7.2.2, the experiments showed that along the axis of the probabilities of the most probable paths, a higher value has a higher accuracy. The basic idea of CRF queues is to build a queue of CRF models, and each model uses the higher probability part to do the prediction. If the probability of a sequence is lower than a threshold t, the data are passed to next model.

If $D(\mathbf{X}, \mathbf{Y})$ is the training set, a filter function is defined as follows, where Y'' is the most probable explanation of X'.

$$r(D,t) = \{(X',Y')|(X',Y') \in D, p(Y''|X') > t)\} \tag{7.3}$$

We define $D' \subset r(D,t)$ as the set of the sequences which are correctly explained. The threshold t^* can be calculated via:

$$t^* \cong arg_t \frac{|D'|}{|r(D,t)|}) = C_1 \tag{7.4}$$

In the equation, C_1 is a selected accuracy higher than the accuracy of the first CRFs in the queue. In order to build the queue, assume the first CRF model is already induced via the algorithm shown in Table 7.2 – we can then use t^* to filter the training set for the next model in the queue. The sequences with a probability of the most probable explanation higher than t^* are removed from the training set. The rests are used to induce the next model in the queue.

$$D_{m+1} = D_m - r(D_m, t^*) \tag{7.5}$$

In our implementation, we simply compute C_1 to remove 60% of the sequences of the first training set. Then, C_1 is fixed for the other models in the queue.

The algorithm of inducing CRF queues is written in pseudo-code in Table 7.3. In each iteration, a CRF model is built; the threshold is computed; and the training set is filtered. The algorithm is run until no CRFs with the required accuracy (C_1) can be induced from the filtered data.

A sequence X can be explained by the queue in the following way. X is explained by the first model p_1 in the queue. If $p_1(Y'|X) > t_1$, where t_1 is the threshold, then Y' is the explanation of the X. Otherwise, X is passed to the second model. If X cannot be explained by any of the models in the queue, the model with the highest accuracy is chosen to explain the sequence. Figure 7.8 shows an example of the CRF queue.

Table 7.3. Algorithm: Inducing CRF Queue from Data

	input: Training Examples $D(\mathbf{X}, \mathbf{Y})$
	output: Learned CRF Queue
	$Q((F_m, \Theta_m), t_m), m = 1...M$
1	$D_1 = D; Q_0 = (\emptyset, \perp)$
2	**for** m=1...M **do**
3	Compute (F_m, Θ_m) on D_m via table 7.2
4	Compute t_m on D_m via Equation 7.4
5	$Q_m = Q_{m-1} \cup ((F_m, \Theta_m), t_m)$
6	Compute D_{m+1} by Applying $((F_m, \Theta_m), t_m)$ on D_m via Equation 7.5
7	**end**
8	return Q_M

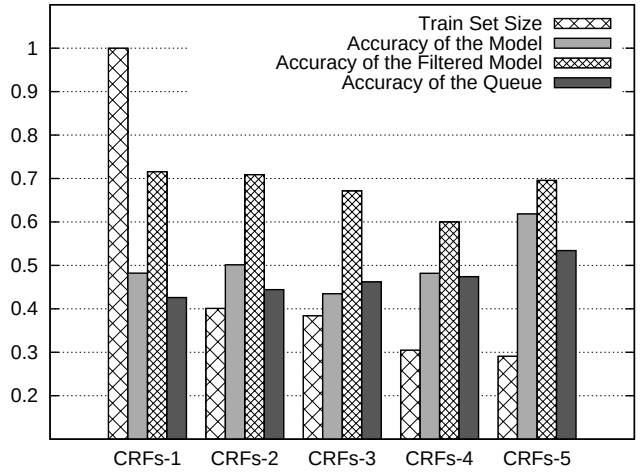


Fig. 7.8. An the Example of the CRF Queue

There are 5 models altogether in the queue. The first column in the figure shows the training set used for the model. In the experiment, each training set consists of 1000 sequences. 1.0 means all of them are used to induce CRF-1. Along the queue, fewer and fewer data are passed to the next model. CRF-5 is trained by less than 300 sequences.

The second column in the figure illustrates the performance of each model on the training set. To our surprise, so many models can be induced with the reduced training sets. Their performances on the training data can be improved by inducing a new set of the features.

The third column is the accuracy of using the model with the threshold t_m, we call it the filtered model. t_m defines the higher part along the axis of the probability of the most probable paths. CRF queues provide a more accurate prediction because the third column is higher than the second one.

The fourth column is the performance of the CRF queue. It is computed by using the current acquired models. For example, in CRF-3, the first, second, and third models build the queue. Along the queue, the evaluations are better and better. In CRF-1, the second column is higher than the fourth one because of overfitting. The performance of the model is better on the training set than on the test set.

In the example, if X cannot be explained with a probability higher than the threshold t_m by all the models in the queue, CRFs-5 should be chosen to explain X. The reason is that its second column is the highest one over the second columns of all the models. To summarize, the third columns of the first four models and the second column of the fifth model are chosen to explain X. Their overall performance, the fourth column in CRFs-5, is lower than any of these columns because of overfitting. From another point of view, the values of the chosen columns are based on the training data; the overall estimation of the queue is the evaluation on the test set.

The **fourth experiment** was designed to evaluate CRF queue. The algorithm shown in Table 7.3 was run on all data sets. The average results over 5 configurations are shown as the third columns in Figure 7.6. The results can be explained as follows.

- The CRF queue outperformed the single model approaches for about 4% on average in all configurations.

When the induced queues are checked one by one, we find that there are a number of queues containing only one CRF model. The main reason is that the first CRFs already has a high accuracy. The second one based on the filtered data is not as good as the first. Therefore, it cannot make any improvements. We summarize this observation thus:

- In the worst case, the CRF queue has the same performance as the single model approaches.

We show the number of models in the queue in the lower parts in Figure 7.7. The results are averaged over the 100 sets in each configurations. The number of models is above 3, which hints that the queue works well in most cases. $\{S1, S3\}$ has a shorter queue because the performances of the single model approaches in these configurations are better, as shown in Figure 7.6.

- The CRF queue is shorter when the single model approaches work better.

7.5 Discussion

In this paper, we constructed a simulation franmework to investigate the issues of inducing features of linear-chain CRFs. The simulation is based on the assumptions that the stochastic processes in the target system can be described as a CRF model. Consequently, we gain a new phase to compare the simulated CRFs and the induced CRFs, and the annotated data set can be automatically generated as many times as required. We designed several configurations for the simulated CRFs, using a large amount of experiments to explore the properties of the learned CRFs. Moreover, we developed two novel methods.

The first one is a feature reduction method that can be integrated into the induction process. The basic idea is to define an estimation to remove a subset of the features from existing ones. The reduction keeps the accuracy of the model and reduces the number of features in the learned CRFs in the experiments.

The second one is an approach based on the properties of the maximum likelihood. A queue of CRF models is constructed for the prediction, which yields a better performance on all the configurations. CRF queues guarantees accuracy no worse than the single model approaches.

The feature induction as well as feature reduction and CRF queue can be implemented in the switching attention learning framework. In Chapter 8, we address how to use this framework to explain the data of table soccer.

8

Explain Data of Table Soccer

By using KiRe, table soccer games can be recoded as a series of data slices. As mentioned in Section 6.1, along the axis of time, each data slice consists of the position of the ball, as well as the position and the angle of the game rods. A data explanation method based on CRFs and SAL was introduced in Chapter 7. In this chapter, an approach is developed to explain the recorded game data using the CRFs method. In Section 8.1, the sensor data and the challenges of explaining these data are outlined. In Section 8.2, the features, the essential elements of CRFs, are defined. In Section 8.3, we address the method using switching attention learning to explain the data of table soccer.

8.1 The Explanation Tasks – an Example

The recorded data can be described as a vector of five variables $(t, \mathbf{x_b}, \mathbf{y_b}, \mathbf{p_r}, \mathbf{a_r})$. The t is a time point at which the data were obtained. $\mathbf{x_b}$ and $\mathbf{y_b}$ are a series of possible positions of the ball, which are given as $2D$ coordinates in a plane. The data are not a single position because of the noises in them. In each time slice, a few possible ball positions are given as the vectors. $\mathbf{p_r}$ and $\mathbf{a_r}$ are the positions and angles of the rods. There are not many noises in the sensor data of the rods. The data of each rod are encoded in these two vectors. Each value in the vector is a real number.

The explanation of the data is to describe a sequence data using skills and concepts. From another aspect, the data are explained as a sequence of actions. We show the explanation tasks in an example. Figure 8.1 illustrates an action sequence. The ball is coming from the midfield of the red team. It is first locked by the attacker of the blue team. Then the player performs a slide-kick action sequence: the ball is slided along the rod; it is then kicked

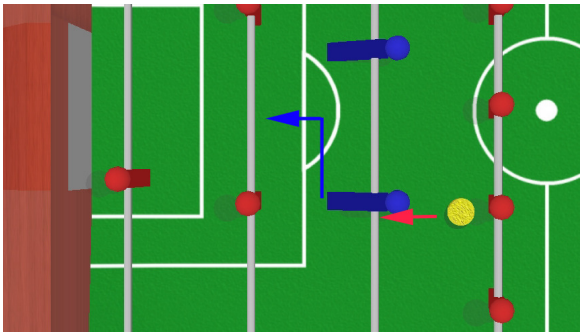
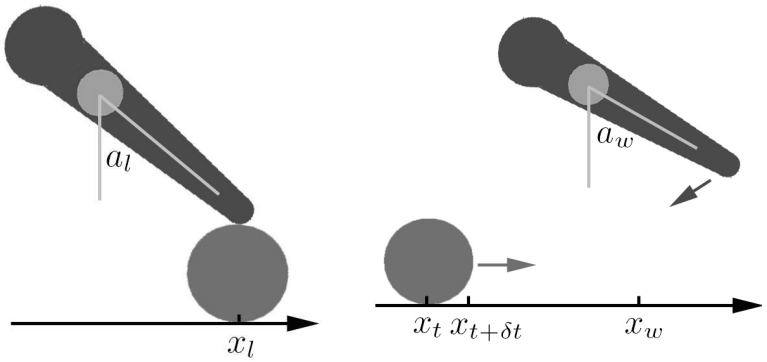


Fig. 8.1. The Action Sequence of Lock and Slide-Kick

towards the goal. In the figure, the arrow lines show the trajectory of the ball. The red and blue colors indicate which team controls the ball.

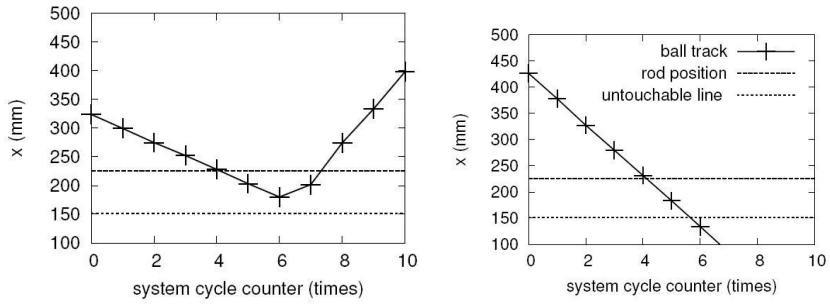


(a)Lock (b)Kick

Fig. 8.2. The "lock" and the "kick" actions

In the example, the ball is first locked by the blue team. The coordinates of the ball, as well as the position and the angle of the rod, indicate a high level state – lock, as illustrated in Figure 8.2(a). The end of the playing figure press the ball towards the playing surface. The ball is stuck between the figure and the surface. Only one slice of the data with the figure and the ball at the proper position, however, is not enough for determining a "lock" state. The "lock" should be a state that can be felt by human beings. In other words, the data in a short time span, e.g. 0.3 second, would be a support for the "lock". As mentioned in Section 6.1, the frame rate of the system is more than $200Hz$.

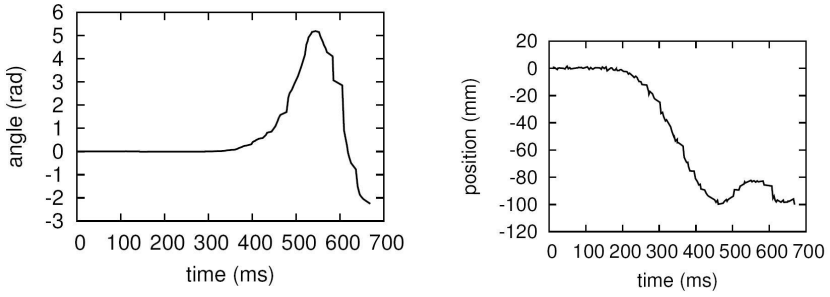
In 0.3 second, 60 data slices are recorded. A "lock" state is observed, if there are 60 data slices indicating the ball is stuck under the playing figure.



(a) A Successful Kick (b) A Missed Kick

Fig. 8.3. The Trajectories of the Ball

In the example, after the ball is locked, a slide-kick action is performed by the player. The state "kick", in which the playing figure touches the ball, takes much less time than a "lock". Although the recording system provide a data slice every 0.005 second, it is still possible that the "kick" state cannot be observed. Therefore, the "kick" is described as a process, as shown in Figure 8.2(b). At the time t, the ball is at the position x_t. After a small time span δt, the playing figure touches the ball. The touch happens at the time point when the ball is knocked out by the playing figure.



(a)Angle (b)Position

Fig. 8.4. The movements of the game rod

Although it is hard to catch the exact time point when the playing figure touches the ball, the process of the kick can be observed from the trajectory of the ball. Figure 8.3(a) illustrates that there are significant changes on the movement of the ball, involving both the speed and the direction of the movement. Figure 8.3(b) shows that if the kick is missed, the ball keeps its previous movements. If a moving window over 10 data slice is used for the observation, a successful kick action should be with the ball trajectory shown in Figure 8.3(a).

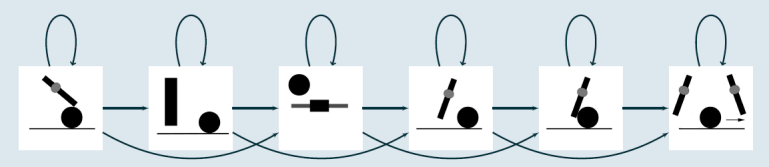


Fig. 8.5. The Action Sequence of Lock and Slide-Kick

In addition to the trajectory of the ball, the data of the rods offer an extra way to understand the action sequence. Figure 8.4 shows the position and angle of the rod along the time. As shown in the figure (a), the rod was first turned to create an accelerating space for the "kick". It was then turned towards the ball with a high acceleration. Figure 8.4 (b) shows the position of the rod, which illustrates a typical slide-kick trajectory. The tilt at $200 - 500ms$ is the slide; the flat at $500 - 600ms$ is the kick. A slide-kick always accompanies the data plot of the rod with a similar shape.

In summary, the explanation should bridge the low-level data and high-level concept such as the slide-kick. In order to explain the data, a model needs to be constructed. The input of the model is the sensor data, the vectors of five variables along the time. The outputs of the model are a series of states, which can be used to describe a certain skill or strategy. Figure 8.5 illustrates such a model. In the model, the discrete states defined in Section 6.2 are used. In the example, these states can eventually be used to describe the high-level skills: lock and slide-kick.

8.2 Encoding Features

In Section 8.1, we assume that the model for data explanation can directly take the sensor data as its inputs. In the implementation, HMMs and CRFs have inputs with discrete values. In this section, a feature encoding approach is implemented to discretize the inputs.

We need the feature encoding mainly for three reasons. First, CRFs have a friendly interface for the features with discrete values. Second, KiRe has a frame rate of more than $200Hz$. Directly processing these data requires a lot of computational power. In the feature encoding, the amount of data can be significantly reduced so that the data can be processed much faster. Third, the encoded features are the stepping stone between the continuous coordinates from the sensors and the final class label such as "lock". The construction of the model for data explanation can thus be easier with the encoded features as its inputs.

Detecting the collisions between the ball and the other objects, e.g. a wall or a playing figure, is crucial for the data explanation tasks. The trajectory of the ball will change significantly if there is a collision. We can approximate the trajectory of the ball by assuming a constant speed and fixed moving direction if there is no collision. With a sequence of collisions along the time axis, a segment can be defined as the data sequence between two collisions. From another aspect, with a sequence of segments, a collision can be defined as a point between two segments. In other words, the data can be explained as a sequence of collisions and segments which appear iteratively.

In a "segment", we assume the ball moves with a constant speed and fixed direction on the $2D$ playing surface. In other words, the trajectory of the ball is assumed to be a *line segment*. If the time axis t is considered, the ball should move along a line in a $3D$ space. The "segment" is therefore a line segment in this space. It can be mapped to two line segments in two planes which have t and x or t and y as their axes.

Hough transformation can be used to find a line-segment in a plane. Each point (x, y) in the original plane can be transferred into a curve in another plane, (r, θ), by using Hough transformation. The relation between the original (x, y) and the transformed (r, θ) is defined as in Equation 8.1. (r, θ) can also be understood as the standard "angle and distance" parameters in the equation of a line.

$$r = sin(\theta)y + cos(\theta)x \tag{8.1}$$

(X, Y) are a set of points , assuming all the points $(x_i, y_i) \in (X, Y)$ are on the same line in the original plane. By using Hough transformation, each (x_i, y_i) is transformed to a curve in the (r, θ) plane. If the (r, θ) plane is discretized with small grids. In the plane, all curves intersect in the same cell. From another aspect, the interpolated line segments in the original (x, y) plane can be found by searching for the cells in the transformed plane; these cells contain intersects of several curves.

The Hough transformation mentioned above is applied to find line segments in the trajectory of the ball. Figure 8.6 illustrates an example, in which the horizontal axis is the time (t), and the vertical is the position along the y direction of the playing surface. The red dots are the sensor data. We use a moving window approach to process the data. The blue lines are the separators of the moving window. The green line segments are the results of the Hough transformation. In the figure, we can find that the line segments can be successfully found; there are more shorter line segments near the "collision" points.

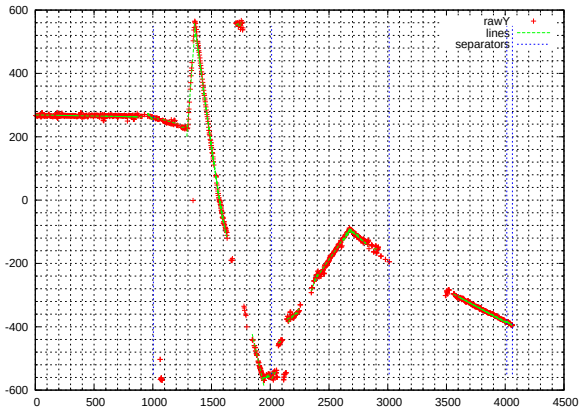


Fig. 8.6. The Line Segments found by Using Hough Transformation

The results of Hough transformation can be used to build the "segment and collision" sequence. Assuming the collision is a kick, the data should support the collision from several perspectives: the first segment and the second one should have very different tilt value; the playing figure and the ball should be very close to each other near the collision point; the ball has a chance to jump up after the kick.

Normally, human players play the attacker and defender in a different way. Further more, advanced players may play the first and the second playing figures of the attacker in a different way. It would thus be useful to identify which rod and which figure is controlling the ball. As mentioned in Section 6.1.1, the laser beam goes through the gap between the playing figure and the field to measure the position of the ball. In real game, the ball, however, jumps very often, especially after a collision. When the ball jumps, the laser beam lost its view on the ball. This information is useful to identify if a collision happens.

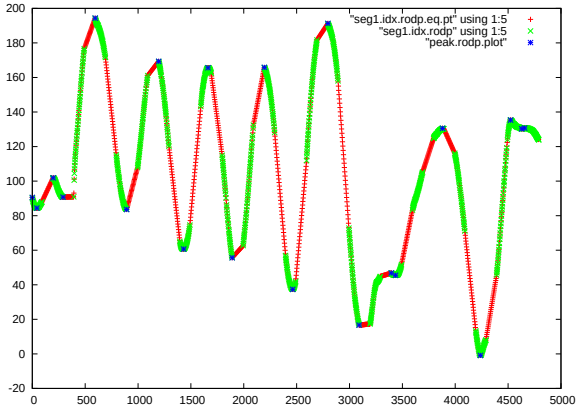


Fig. 8.7. The Segments of the Rod Position

Similar to the trajectory of the ball, the movement of the rods can be analyzed. Using a local search algorithm, the peaks, shown as blue dots in Figure 8.7, can be found. The segments of the rods can be encoded as features. These features can be used to support the actions such as a "kick".

There are all together 56 features, in which the domain knowledge, the spatial relations between the figure and the ball, and the results of the Hough transformation are integrated. All these features are encoded as booleans, each of which has two values, true or false. Some of the features can of course be encoded as integers. We choose boolean to avoid the bias among the encoded features. Consequently, all features are equally important in the high-level model.

8.3 Explain Recorded Data Using CRFs and Switching Attention Learning

By using switching attention learning, several learners can work together for a systematic learning. Such learning system can be constructed for the data explanation tasks. In this section, we address the construction of the learning system for the data explanation using CRFs and SAL. There are several modules in the system: segmentation, hypothesis computation, training set generation, and several learners using CRFs and SAL.

The sequential data have the form (X, Y), as mentioned in Section 5.2.1. $X = (B_1, B_2, ..., B_I)$; B_i is a vector of the conjoint of the boolean features. In Section 8.2, 56 such features are encoded. The longer the sequential data are, the more computational power is required for training the CRFs. A seg-

mentation algorithm is thus required for dividing the longer sequences into shorter ones.

The segmentation algorithm divides a long data sequence into several smaller ones. In the playing field, there are several areas where the ball can hardly stay, e.g. the corners or near a wall. In these areas, however, the noises can easily be generated. When the ball is out of the table, the recognized ball position would be in these areas. The segmentation algorithm can be built according to this observation: after the recognized ball position stays in the high noise areas for a particular time span, the data can be separated into two segments. A long sequence of the data can thus be divided into several smaller ones.

In the sequential data (X, Y), Y is a hidden state which can also be understood as a hypothesis. In the data explanation tasks, the "hypothesis" is defined as a place at which a collision would happen. The hypothesis has four possible values for different situations. The first situation is that it marks the exact place the collision happens. The second situation is that it is not a collision; the two segments before and after it can be merged into one. The third situation is that the place of the hypothesis can be used to compute the end of the segment before the hypothesis. The forth situation is that the place of the hypothesis can be used to compute the start of the segment after the hypothesis.

The hypotheses can be computed automatically. Three methods are integrated for this computation. The first one is developed by considering the "touch" relation. A hypothesis is created if the ball and the playing figure touch each other. The second method is developed by considering the "jumps". If the ball cannot be observed for a certain time span, a hypothesis is created. The third one is developed by using the local search algorithm illustrated in Figure 8.7, where the search algorithm is for the game rods. Here, the search is used for the trajectory of the ball. The hypotheses are created at the places where local peaks can be found by the search algorithm.

In order to create the training set for CRFs, we need to manually annotate the sequential data. The annotation of the data is to find out where the collisions are, given the recorded data. In other words, we need to use a series of start-end connected line segments to approximate recorded data. The red and green dots in Figure 8.8 are the recorded data. By considering these points, we can easily annotate the data using the line segments. These annotation are regarded as the ground truth, and are used to compute values for the hypotheses. Annotating data is very time consuming. We annotated about 200 sequences in about 80 hours.

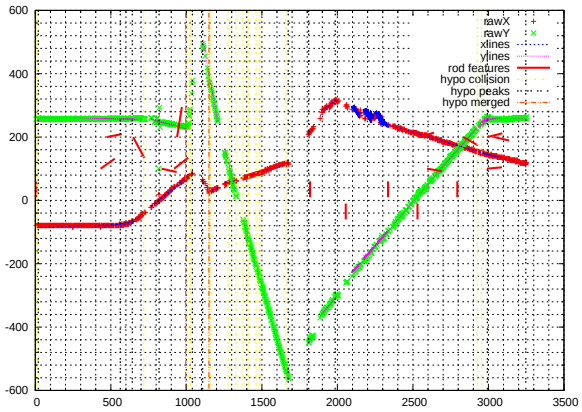


Fig. 8.8. The integrated Information of a Sequence

We integrate all known information in Figure 8.8. There are three types of hypotheses, collisions, peaks, and merged ones. The merged hypotheses are created because several hypotheses are close to each other along the time axis. It could happen that a local peak, a jump, and a "kick" happen within 10 data slices. The information of the active rod is shown in the figure as red line-segments. The vertical red segments are the turning of the rods. The segments with angles $a > -90^o$ and $a < 90^o$ refer to the movement of the active rod. The active rod is the rod that controls the ball. The value of an angle a indicates the speed of the movement.

The training set of CRFs can be generated by considering the generated hypotheses and the ground truth. Given the annotated information, the values of the hypotheses can be computed. And each feature in X can be calculated given the hypotheses.

With the encoded 56 features and the feature induction algorithm mentioned in Chapter 7, we can induce the features step by step. A switching attention learning system with the following three learners are developed. The CRFs model is trained each time after a new set of features are induced and reduced.

- Feature induction of CRFs
- Feature reduction of CRFs
- Training of CRFs

An experiment was designed for this learning. The annotated 200 sequences are used as the training set. We develop a CRFs model with five hand-coded feature functions. This model is trained. Its performance is re-

garded as the base line for the evaluation. We built the algorithm to interactively induce the features. In each iteration, 3 most promising features are induced. The training set was divided into 31 subsets. One set is used for the evaluation, the other 30 sets are used for the training. The training of the CRF model is evaluated over the training set.

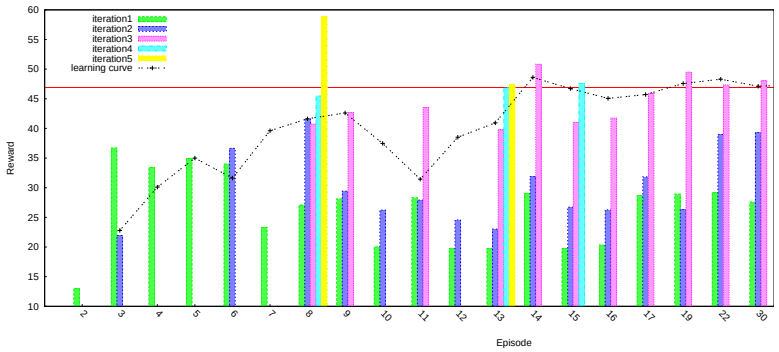


Fig. 8.9. The Learning Curve

The results of the experiment are shown in Figure 8.9. The red horizontal line shows the performance of the hand-coded model. The horizontal axis is the size of the training set. The vertical is the performance of the model on the evaluation set. The hand-coded CRFs can predict with a correct rate of 78%. The performance of the model in each iteration in the feature induction is shown as the color columns. Several iterations are involved at each evaluation point. The black dot lines show the learning curve. The best performance of the model is acquired by considering 8 subsets of the training data. When more subsets are considered, from 13 to 30, the models acquire the stable performance close to that of the hand-coded model, which indicates the feature induction is successful in the experiment.

Although the learning curve supports the improvement of performance, the results, however, is not very useful. An accuracy of about 78% can only produce a coarse explanation of the data. There are mainly two reasons for this problem. One reason is that much more annotated data are required for the training. In the simulation, the size of the training set can easily reach 1000. The annotated data of table soccer, nevertheless, are hard to be obtained because of the amount of work required. Another reason is that we used several thresholds in the computation of the features. The value of a feature can be oscillated when it is near a threshold. In the future, we would investigate semi-supervised learning to combine both labeled and unlabeled data.

Part III

Tetris

9

Outline

In this part, first we introduce the background information: the artificial players of Tetris in the literature are reviewed in section 10.1; bandit based Monte-Carlo planning (UCT) is introduced in section 10.2; learning by imitation and support vector machines are explained in section 10.3. We developed two artificial players for Tetris. In the first player in Chapter 11, the game is modeled as a planning problem using UCT. In the second player in Chapter 12, support vector machines are implemented within the SAL framework for playing Tetris.

10

Preliminaries

Tetris was first invented by Alexey Pajitnov et al. in 1984, and remains one of the most popular video games today. It can be found in many game consoles and several desktop systems in PC, such as KDE and GNOME.

Tetris is a stochastic and open-end board game. A piece of block is dropped from the top of the board. The piece is randomly chosen from seven predefined ones, and it falls down step by step. The player can move and rotate the current piece to place it in a proper position. A new piece appears at the top of the board after the current one touches the ground. A fully-occupied row will be cleared and the blocks above it will automatically fall down one step. The goal of the game is to build as many such rows as possible.

In the two or more players' competitions, if one player has removed $n(n > 1)$ rows in one turn, all other players will receive an attack of $(n - 1)$ rows of blocks, adding to the bottom of their game fields. Each attack row would contain $(n-1)$ empty cells in random positions. Thus removing multiple rows in single turns brings even more benefits than rewards. Highly skilled human players prefer to plan and remove three or even four rows using a single falling piece, while beginners and many of the existing tetris artificial players tend to remove rows as soon as possible in each turn to survive the game. The game is over when only one player is still alive in the competition, and of course the last player is the winner.

10.1 Tetris AI players

The single Tetris game has been used as a test-bed in the research in artificial intelligence. Researchers developed artificial players using different approaches [70]. Fehey created a hand-coded player [31], Böhm et al. employed genetic algorithms [55], and Szita et al. used cross-entropy methods in Tetris

[3]. These players can play the single game, clearing hundreds of thousands of rows, which would take several weeks or even months for a human player.

Some methods are successful in playing single Tetris games, some are not. For example, the standard 10×20 Tetris game is still a challenging task for the methods in reinforcement learning [30] [18]. The number of rows that a player can clear is widely accepted as a criteria for the evaluation. So far, several successful artificial players e.g. in [55], [3], and [31], are based on building an evaluation function with linear combinations of the weighted features. These features were listed in [70].

The competition in Tetris is certainly an interesting topic. In theory, the two-player Tetris is much more complex than the single one [59]. Assuming both human and the artificial player handle the piece with the same speed, human players can defeat the best artificial player with ease in the competition mode. To our knowledge, the existing artificial players cannot create many attacks in the competitions. The researchers evaluate their players mainly in single games.

10.2 Bandit Based Monte-Carlo Planning

Planning is a branch of research in artificial intelligence. States, transitions, and a search method are three basic elements in planning. One state can be transferred into another state via a transition. Final states are the special states which associate with results. The objective of the planning is to search for a sequence of transitions that can lead a start state to the preferred final state. The search method guides the process of searching.

Bandit-Based Monte-Carlo planning (UCT) [43] is a planning method that integrates a bandit algorithm, a Monte-Carlo method, and Markov Decision Processes (MDPs). The process of planning is consists of a number of "roll-outs". The planning starts at a state. In a rollout, the start state is regarded as a "root". The algorithm chooses a transition to proceed to the next state. The process will be continued until a fixed number of transitions (depths) is reached. At the end of the rollout, the algorithm will update its parameters, the next rollout will be started again at the "root" using the new parameters.

The parameters are updated in the following way. Starting from the beginning of one planning episode, the number s of visits of a state in the search tree and the number t of visits of each action of the state and the piece are constantly updated according to each selection of the actions. According to the algorithm UCB1 [4], the action selection is based on the upper confidence

bound in track of the immediate reward and bandit score of every arm (action) of the bandit machine (game state) as:

$$I = \underset{i \in \{1,\ldots,K\}}{\arg\max} \{R_i + c_i\} \tag{10.1}$$

where R_i is the immediate reward from performing the action i, and c_i is a bias sequence chosen as:

$$c_i = \lambda \sqrt{\frac{\ln s}{t}} \tag{10.2}$$

where λ is a constant factor manually chosen for balancing of the exploration-exploitation trade-offs. Higher λ values result in higher chances of randomized explorations based on the bandit scores, while smaller λ values lead to greater possibilities of selective exploitations according to the immediate rewards.

The UCT algorithm works in the following way. First, A fixed amount of computational power is allocated to each step. Then, the allocated computational power are used to run the rollouts as many times as possible. Next, the algorithm choose the transition at the root node which is best evaluated by Equation 10.1. Finally, the chosen transition is used to update the root node; the algorithm can start a new iteration using the new root.

10.3 Learning by Imitation and Support Vector Machine

Imitation is essential in social learning [6]. Assuming the similarities between the observations and themselves, humans acquire various skills via imitation. Imitation learning can be applied in robotics and automatic systems in several ways[17]. For instance, Billard et al. built a system according to the structure of the human brain [1]. Atkeson et al. developed a method to explain the actions of a demonstrator, and to use the explanations in an agent [19].

Learning by imitation has been widely applied in robotics, especially in humanoid robots [1]. The core idea of imitation is to improve the similarity between the imitated system and the imitator, even if certain physical or virtual dissimilarities exist. In this chapter, a framework is developed to imitate both human and artificial players. The structure of our approach is certainly different from human brains or the models of the other artificial players. Generally, we follow the idea of learning by imitation. To our knowledge, it is the first time that imitation learning has been applied in Tetris.

Support Vector Machine (SVM) was first proposed by Cortes and Vapnik in 1995[23], and became an important method for data classification. SVM is well-developed. I was implemented in several open source packages which were available in Internet. In this paper, SVM is used as a tool. Our implementation is based on LIBSVM [20]. We modeled the imitation tasks in Tetris as a standard data classification problem which can be finally solved by SVMs.

Incremental learning is mainly about a series of machine learning issues in which the training data is available gradually [2]. It is a special learning method with which a certain evaluation can be improved by the learning process during a fairly long period. In order to do that, we defined a learning paradigm: switching attention learning [25]. In the paradigm, there are multiple learners with their inputs and outputs forming a loop. The performance of one learner generates potential improvement space for the others. Following this idea, Tetris is used as a test-bed. Our artificial player can choose a game played by a stronger player as its target to imitate.

11

Playing Tetris Using Bandit Based Monte-Carlo Planning

Yet most of the existing artificial players known are based on a planner for only one- or two-piece. This paper was motivated by creating an artificial player based on the planning of a long sequence of pieces. We modeled our player in Tetris planning problem with the Monte-Carlo planning method. In order to balance the exploration-exploitation trade-offs in the planning process, we employed the bandit algorithm to guide the planning process. As for state revisiting, we created a method to store the visited game states in a specially designed database. We also created a hash function to quickly locate and operate the information of a given game state in the database. In order to reduce the branching factor of Tetris planning, we created an intuitive evaluation function and combined it with the UCT algorithm.

The highlights of this paper can be summarized as follows:

- We modeled the artificial player of Tetris using the UCT algorithm.
- Our method of the database of the visited states provided support to UCT and improved the performance of the planner.
- By pruning the planning tree, the player can defeat the artificial player developed by Fehey, which is regarded as the benchmark.

This chapter is structured in the following manner. First, in Section 11.1, the related works are overviewed. Then, in Section 11.2, we present our solution on modeling the tetris planning problem with the bandit-based Monte-Carlo planning method. Our method to design the knowledge database and store the information of the visited game states is presented in Section 11.3. The idea of combining the evaluation function to the UCT algorithm is discussed in Section 11.4. The experiments and the results are shown and analyzed in detail in Section 11.5. In the final Section 11.6, we draw the conclusion and discuss the future work.

11.1 Related Works

To create an artificial player for a board game, the general components are the search method and the evaluation function. The board games which are solvable by brute-force methods, such as Othello, have already been dominated by game programs using various search methods, such as LOGISTELLO [16]. Board games such as Checkers are solvable using knowledge database combined with search methods, one such example is the program named CHINOOK [63]. Many board games, e.g. Chess and Go, are currently unsolvable, thus are still challenging tasks for artificial intelligence researchers. To improve the performance of the artificial players for these board games, one of the tasks for the researchers is to balance the trade-offs between the search depths and evaluation functions [10].

The Monte-Carlo planning method (MCP) has offered a new solution to artificial players of board games. In 1993, Bernd first modeled the board game Go with the MCP algorithm [15], and his Go player had a playing strength of about 25 kyu[1] on a 9×9 board. Soon the MCP method was successfully applied in other board games, such as Backgammon[49]. In 2006, Levente Kocsis and Csaba Szepesvri developed a new search technique named UCT, which stands for *Upper Confidence Bound applied to Trees* [43], and proved that UCT to be more efficient than its alternatives in several domains. Instead of uniform sampling of the game actions, UCT uses the multi-armed bandit algorithm to guide the action selection of the planning process. Later applications using the technique, such as MoGo[2], demonstrated that this technique can be successfully applied to the game of Go.

Learning techniques have also been applied to improve the perfomance of artificial players of board games. The first such approach was the one by Samuelson in 1959 [61]. He was able to show how a program can learn to play Checkers by playing against itself. In 2010, Takuma Toyoda and Yoshiyuki Kotani suggested the idea of using previous simulated game results to improve the performance of the original Monte-Carlo Go program [73], and their work announced positive results on the larger Go board. In Tetris, Böhm et al. used genetic algorithms for the heuristic function, and our previous work had introduced learning by imitation to the artificial player of multi-player Tetris games [55].

Yet to the best of our knowledge, UCT has not been applied in artificial players for Tetris.

[1] In Go, the rank of $30 - -20$ kyu refers to a *Beginner* level.
[2] Website: http://www.lri.fr/ gelly/MoGo.htm

11.2 Planning Tetris Using UCT

In this Section, we discuss how we model the Tetris planning problem using the UCT algorithm.

There are two possible values for every cell in the game field, e.g. occupied and unoccupied, so the standard Tetris search space consists of 2^{200} game states. The branching factor is 162 for a given game state without the piece information, which is the sum of all possible placements of actions from the 7 different pieces. The large branching factor brings us to the idea of using the Monte-Carlo planning method in our solution to the artificial Tetris player. The core feature of the Monte-Carlo planning is to sample as many future states as possible from all actions of the given state of the game for a certain period of time, and for each episode evaluate only the leaf state using a fast evaluation function. In the end, the algorithm takes the action with the best evaluated reward in the planning as the result of the algorithm.

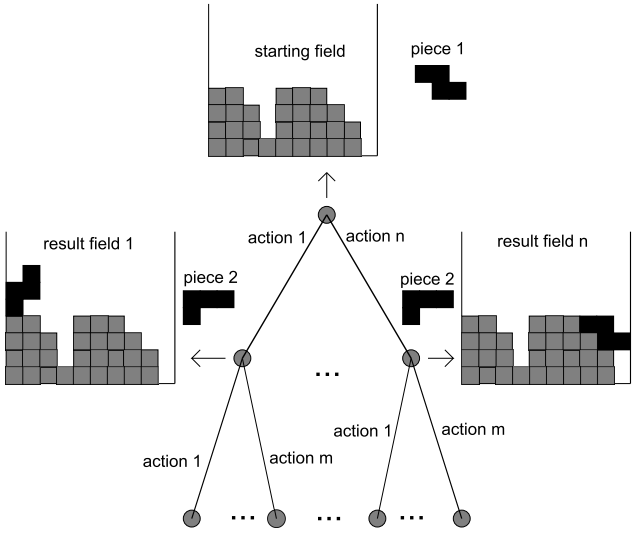


Fig. 11.1. Node of game field state and piece in planning tree

A sample of the planning tree is shown in the Figure 11.1. In our model of the Tetris planning, we consider each state of the game field, together with a given piece, as a single node in the planning tree. For instance, the root node consists of a game field, which is the rectangular area with gray square

blocks inside, and a piece, in this case a ”Z” shaped piece displayed by four black square blocks. The fields in the nodes are the so-called ”cleared fields”, which means that no fully occupied, removable rows are contained in such fields.

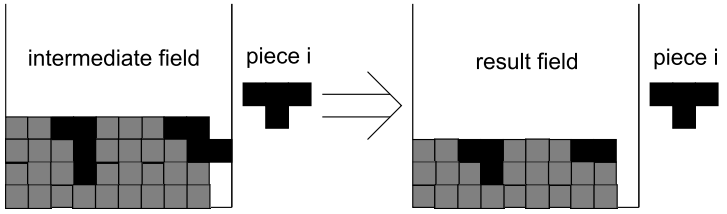


Fig. 11.2. Procedure of removing fully occupied rows

The paths through the planning tree represent the actions associated to the given piece. By following the path of one starting node, the piece is added to the game field according to the index of the encoded action, and a target field is generated. The field will then be checked for removable rows, and if there are rows removed in the field, such rows are removed and a reward will be given according to the predefined game rule. This procedure is described in Figure 11.2. The fully occupied row is removed from the intermediate field, and then the resulted field and the piece i form a new node in the planning tree.

11.2.1 The Structure of the Planner

The pseudo code describing our planner is displayed in Algorithm 1 and 2.

In the beginning of the planning episode, the state of the game field and the sequence of pieces are the input parameters of the planner. The planner initiates the growth of planning tree and starts planning phases. The rank of the paths in the planning tree are calculated directly by using the rewards

```
   Input: state, list of pieces
   Output: action
1  initialization;
2  while not time out do
3  |  search(state, first piece);
4  end
5  action←selectBest(state, pieces);
6  updateTree();
7  return action;
```

Algorithm 1: One Planning Episode : Function **doPlanning**

gained from performing the action associated and removing the fully occupied rows in the field. Removing rows in a single turn has certain rewards, which in our approach is given as 0.1 for a single row, and 0.3, 0.6, 1.0 for two to four removed rows. This encourages the players to remove multiple rows instead of only one row in one turn. The planning episode continues to run as many phases as possible until a certain time out rule is reached, and returns the action with the highest reward in the root node of the planning tree. The subtree of the node following the path of the selected action is preserved for future planning, and other nodes are deleted at the end of each planning episode to reduce memory consumption.

```
   Input: state, piece
   Output: reward
1  type←stateType(state, piece);
2  switch type do
3  |  case type == normal node
4  |  |  action←selectAction(state, piece);
5  |  |  state,reward←performAction(state, piece, action);
6  |  |  updateTree(state, piece, action, reward);
7  |  |  return reward;
8  |  case type == leaf node
9  |  |  return 0;
10 |  case type == terminal node
11 |  |  return -1;
12 |
13 endsw
```

Algorithm 2: One Planning Phase : Function **search**

In each planning phase, the planner selects and performs one of the actions for each piece in the given sequence. Each performed action results in a child

node in the planning tree, together with a reward accordingly. Once all the pieces in the given sequence have been performed with an action, and the leaf node is reached, the planner sums up the total reward gained in the current search path, and updates the reward information backwards from the leaf node to the root.

In Tetris, a common game state usually does not have any information on the winning chance. To simplify the recursion on the leaf node, we consider it to be with zero reward, which means such nodes do not have any influence on their parent nodes in the search path. One exception is that, if a node contains a state of the field which is by Tetris rule a terminal state, the reward is always set to a big negative value. Such behavior makes the actions leading to terminal states less likely to be chosen by the planner.

The method for sampling the actions is the key feature of the Monte-Carlo planning. Comparing to the traditional uniformed and randomized sampling methods, the multi-armed bandit algorithm has advantages in balancing the trade-offs between explorations and exploitations during the planning process, and is proved to be more efficient than other methods in many domains[43].

11.2.2 Bandit Algorithm

In our method, we consider each state of the Tetris game field together with a given piece as a separate K-armed bandit machine, where K is the number of possible actions for the piece given.

In the function of the action selection, first the bandit score of each action of the given piece is calculated by using the visiting information of the node in the search tree, then the immediate reward from performing the action is returned. The sums of the bandit scores and immediate rewards are used to rank all these actions. The action with the highest sum is chosen to be the return of the function. If there are multiple actions with the same highest sum, the result is chosen randomly from the list of such actions. Notice that the formula 10.2 will be invalid for the nodes of the search tree where some of the actions are never visited before. For such nodes, an action is selected randomly from all of the unvisited actions.

In the standard UCT algorithm, the search tree is updated and pruned, and only the sub-tree of the state from the selected action will be kept for the next planning episode. For board games where game states are less likely to be revisited in the future steps of a single game, such behavior would have little influence on the future planning process. But the state revisiting happens quite often in Tetris because of its game rules. Therefore, the information of

the explored states in previous planning episodes would play an important role in the Tetris planning. In the next Section, we will discuss Tetris state revisiting.

11.3 State Revisiting

Before designing our database for the visited game states, we think of the information that is useful for the future planning episodes. First, we want to start each planning episode of a root node from scratch. So the information of the number of visits to nodes and actions is not to be stored, because such information is a bias to the given sequence of pieces of the previous planning episodes. The immediate rewards and targeting states of the actions associated to one node can be easily and fast computed in the planning phases, thus the information can also be ignored.

In our method of planning, a node in the planning tree is made of a state of the game field and a given piece, and the information of the node consists of the following components which is necessary to be stored:

1. The highest reward over all the actions, and
2. The highest reward of each action.

The information of the Item 1 is the key to our idea of storing and reusing the information of the explored game states, because it represents the summarized results of its associated previous planning episodes, and can be easily combined with the results of any future planning episodes. The information of the Item 2 can be abstracted to a list of actions that matches the highest reward, which can be combined easily with the future planning results.

Considering the planner of the artificial player plays 100 pieces in a single game, and for each planning episode, the planner explores $1,000$ phases. Then the total number of explored states of a single game is approximately $100,000$. Assume that one quarter of these explored states are revisited states, then in the end there are $75,000$ newly explored states in a single game that need to be stored. This is not a big amount of data, but the time cost and space consumption for such information would be the bottle neck of the planner. In our approach, we store only the information of the root node in every planning episode. This way, the size of the stored nodes is significantly reduced, while the most useful information of each planning episode is preserved, and the time cost for storing and retrieving such information is kept low.

Now that we have our information stored in a database, the final task is to find a fast and easy way to load and save such information. In the following Section, we will present our design of the database using hash functions.

11.3.1 Hashing in Database

Although not all of the game states will be explored and stored in our approach, locating a specific state in a huge amount of data is still not an easy task. One of the possible options is to use an existing database management systems. However, such systems are inappropriate for our approach, as the data we want to store are small in the single size and have little relation to each other.

In our early approach, we tried to store the data in a single file, which is easy to implement. But locating the data of a certain game state is not an easy task. One of the possible method is to use the "sparse file" system for storage, and every state is stored to a certain position in the "sparse file", where positions are calculated using a hash function. Since there is an "offset limit" in the size of a "sparse file", it is difficult to create a perfect hash function to generate positions for the 2^{200} states in Tetris without collisions.

Another option is to use rather a simple file system with each state hashed to a specific file. Like the "sparse file" system, the simple file system is also dependent on the operating system to locate the entry of a certain file. The difference is that the hash function can easily be created for the simple file system. Also because game states are stored in separate files, there will be no "offset limit" of a single file, and thus the collisions of positions are easy to be avoided.

For any file system, the key issue is to balance the number of files, the number and depth of folders, and the size of each file containing the data. Too many files or subfolders in one folder could cause more time for the operating system to locate the entry of the target in the disk. The size of each file directly affects the computation time to store and retrieve the data for the program.

In our approach, each state of the game field generates a file name directly according to the value of the field encoded by a vector of integers. In this way, every state would have a unique file name, and the collision problem mentioned above is solved. Another advantage is that, the data of the game field is hidden inside the name of the file. And from another point of view, this method reduces the size of the storage.

Then, all such files are separated into different folders in a folder tree of 5-depth. In each depth of the folder, there are up to 16 subfolders. This

scattering method is used to avoid too many subfolders or files in a single folder. Our later experiment on random sequences of pieces showed that for the Linux operating system the computational time had less than 1% differences in runtime from the beginning to the end of the experiment, where the number of the saved states in the database increased from zero to $100,000$.

```
Input: state, pieces
Output: action
initialization;
while not time out do
    search(state, pieces, 0);
end
combineKnowledge(state, pieces);
action←selectBest(state, pieces);
updateTreeEx();
return action;
```

Algorithm 3: One Planning Episode : Modified Function **doPlanningEx**

In each planning episode, the planner starts planning from scratch, using only the information of the game state and piece and current planning tree. Then after the planning phases are completed, the planner loads the information of the root node in the planning tree from the database. Such information is combined with the newly explored information in the planning tree. The combination rule is simple. If the highest reward of the root node in the database is bigger than in the planning tree, the information in the database will completely override the information in the planning tree, and vice-versa. If the two rewards are the same, then the list of actions matching the highest reward will be merged. Algorithm 3 shows a modification to its previous version discussed in Section 11.2.1.

11.4 Pruning the Planning Tree

The previously introduced method is based on the sampling of all possible actions of the given piece in the given game state. However, many of the actions are not worth exploring, because they often lead to useless or even bad game states. In this section, we created a method to prune the planning tree to reduce the number of actions that need to be sampled, and thus to improve the performance of the developed player.

From records of many Tetris games played by artificial and human players, we studied that one of the most important features of the proper placement of a given piece in the game field is to avoid creating holes in the field. A hole in the game field is defined as an unoccupied cell that is covered by one or more occupied cells over its top. A row in the game field containing any holes cannot be removed, and would cause the player to spend more time to deal with.

Since many actions sampled by our method would create such holes, we created a method to prune these actions from the planning tree. The pruning is based on the increment of holes from the original game state to the resulted game state by performing an action. In addition to the reward and the bandit score of each action discussed in Section 11.2.1, the number of holes created by the action is considered to be a negative effect on the sum of the former two parameters. The Equation 10.1 is then modified as:

$$I = \underset{i \in 1,\ldots,K}{\arg\max}\{R_i + c_i + P_i\} \tag{11.1}$$

where P_i is a negative value defined according to the number of holes created, and is defined by the following equation:

$$P_i = \gamma H_i, (-1 \leq \gamma \leq 0) \tag{11.2}$$

where H_i is the number of holes created by performing the action i to the game field, and γ is a negative factor according to the number of rows removed from the result game field. The reason for γ being different is that, unlike the actions that can only create holes in the field, the actions that can both remove multiple rows and create some holes may still lead to a good game state, and thus should not be totally ignored.

11.5 Experiments

We have conducted three experiments for our artificial Tetris player.

The first experiment was meant to test the validity of our method. In order to measure the performances of the developed player when using the database of the visited game states to support the UCT algorithm, we started the experiment on an empty database, and let the player repeatedly play Tetris on a fixed piece sequence from the start of the game till the end. Two of the game parameters are to be evaluated: a) the final score of the game, and b) the ratio of the roll-outs in one planning episode comparing to the standard UCT algorithm. The former parameter stands directly for the performance of

the developed player, and the latter indicates the effectiveness of using the database of the visited game states in the future planning process.

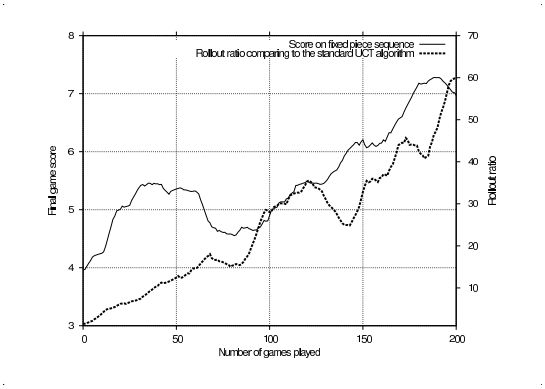


Fig. 11.3. Experiment on a fixed piece sequence

The results of the first experiment are shown in Figure 11.3. The solid line in the figure displays the final score of each game. The game score is based on the sum of the number of removed rows in each turn. We use the same scoring rules for the rewards of removing rows in the standard tetris. We can see that the final score grows as the number of played games increases. This shows that our method to store the information of the visited game states and reuse it in future planning process can successfully support the standard UCT algorithm and improve the performance.

Comparing to the standard UCT algorithm as a basis for the number of roll-outs per planning episode, the roll-out ratio of each planning process with the support of the database of the visited game states is shown in the figure with the thick dashed line. The number of roll-outs is piled up when the state is revisited in the future planning episodes. The results show that the knowledge database helps the standard UCT algorithm to do more roll-outs in the planning process when the states are found revisited in the database, and the performance is hence improved.

One observation in the experiment is that, although the trend of the two results is going in a growing manner, there exist some falls of scores and ratios at some point of the experiment. After analyzing this phenomenon, we found out the reason is that at some point of the game, some newly explored states produced some immediate rewards which are higher than those of the states of the previous planning episodes. This resulted in the change of the choice of the actions for the piece at the point of the game, and lead to some

future states which are brand new to the player's knowledge database. We can also see in the figure that after some more explorations of the games, the final scores soon went up again.

The second experiment was designed to analyze the total size and the revisiting state coverage percentage of the database of the visited game states. Unlike the first experiment on a fixed piece sequence, we let our player continually playing the tetris games with randomly generated piece sequences. The main idea is to let the player meet and explore as many unvisited game states as possible, while examine the percentages of states revisited in games with completely different piece sequences.

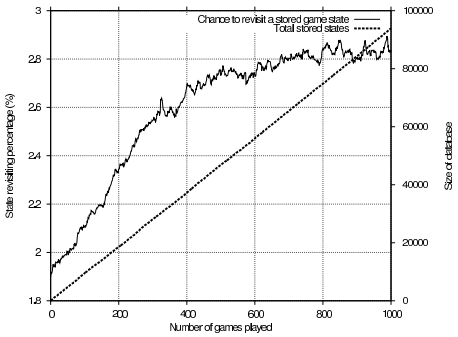


Fig. 11.4. Experiment on random piece sequences

From the results of the second experiment displayed with the dashed line in Figure 11.4, we can see that the total size of the stored game states in the knowledge database is growing with a constant factor in our case. We thus can conclude that given randomly generated piece sequences, the player is able to increase its knowledge of the Tetris game.

Shown with the thick solid line in the figure, as more visited game states are stored in the database, we can see that the percentage of state revisiting is increasing. This means that is more likely to revisit a previously explored game state, even when the game has a completely different piece sequence. On the other hand, the percentage of states revisited during games with different piece sequences is still low, which means that currently the size of the database is not big enough to cover many useful Tetris game states.

Combining the results of the first two experiments provides evidence that our artificial Tetris player can successfully play the standard Tetris game using the UCT algorithm, and has the ability to learn from the played games

and improve its future performance with the help of the knowledge database of the previously visited game states. By repeatedly playing the Tetris games using randomly generated piece sequences, the performance of the player will improve, as more useful game states will be covered by its knowledge database.

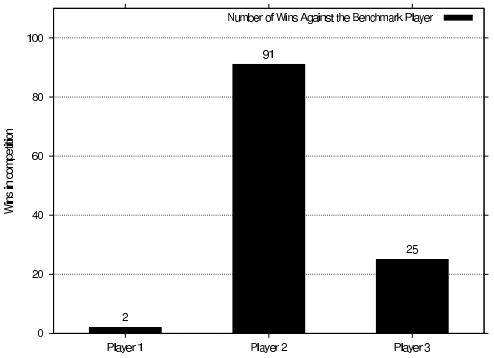


Fig. 11.5. Competitions between players

The third experiment was the competitions against Fahey's benchmark player[31] using different approaches. The winning percentages are calculated on a basis of 100 rounds of games, and the results are shown in Figure 11.5. The three columns represent the different approaches used to develop the player: 1) the UCT based player with only the hashing database, 2) the UCT based player with both the hashing database and the pruning method, and 3) the SVM based pattern player in our previous work[87].

As we can see in the figure, with only the hashing database, the player's performance against the benchmark is very low. With the support of the pruning method, the performance of the player is significantly improved, as shown by the number of wins in Figure 11.5 increasing from 2 to 91 over 100 games, and is competitive comparing to our previous work of the SVM based pattern player.

There is a trade-off in including pruning in our approach. The complexity of the included method affects the efficiency of the planner in both the action selection and the roll-out sampling. According to the statistics of the experiments, the number of roll-outs per planning episode dropped by $1/2$ when the pruning method is included, while the number of pieces successfully played per game raised by 25 times. Conclusion can be drawn that such method is

suitable in our approach to improve the performance of the developed player.

11.6 Discussion

In this paper, we have developed an artificial Tetris player using the bandit-based Monte-Carlo planning method. Different from many existing artificial Tetris players, our player is built on the ten-piece planner. Our idea is to use the Monte-Carlo planning method to sample the possible actions of the given pieces in the game field, and use the bandit algorithm to balance the exploration-exploitation trade-offs and guide the planning process.

One of the key challenges of our work is to find a good solution to make use of the information of the visited states during the planning process, as such information is not kept and reused in the standard UCT algorithm. We created a method to store the information of the visited game states in a specially created database file system. The information can be loaded and reused in the future planning episodes when the states are revisited, and the scheme provides our artificial player with the learning ability.

The high branching factor causes the planner to spend much of its time exploring possible actions, while many of such actions are useless and often lead to unwanted game states. We created a method to prune the planning tree during the planning process to reduce the number of actions to be explored, and eventually improve the game performance of our player.

The experiment results show that our player can successfully play the Tetris game. By using the stored information of the visited game states as a support to the UCT algorithm, the results of the experiments show that the performance of our player improves as the number of games played increases. The player could explore the unvisited Tetris game states using randomly generated piece sequences and improves its game performance. With the pruning method, the developed player has significantly higher chance to win a multi-player Tetris game in competition against the benchmark of the Tetris players.

The results of our second experiment on randomly generated piece sequences showed that our database has not yet covered a high percentage of the useful game states. In the next step, we will continue the experiment on exploring unvisited game states for the database, and analyze the use of larger database in the Tetris games.

Currently we use an intuitive method to prune the planning tree. Although the overall performance of the developed player is improved, careful studies are needed to analyze the trade-offs between more complex pruning methods

and the changes in the player's performance. This is another interesting topic be in our future plans.

12

Playing Tetris Using Learning by Imitation

This research was motivated by building an artificial player for the competitions in Tetris. As a human is superior in the competitions, we employed learning by imitation to clone the game skills of human players. The highlights can be summarized as follows:

- We developed an open source platform for the competitions.
- To our knowledge, learning by imitation is novel in Tetris.
- Our artificial player can acquire diverse game behaviors by imitating different players.
- Our player has chances to defeat the best-known artificial player in the competitions.
- The framework supports incremental learning.

This chapter is structured in the following manner: first, the construction of a open source platform is introduced in Section 12.1. Then, the encoding of the features is addressed in Section 12.2. Next, an approach using support vector machines is developed for playing Tetris in Section 12.3. After that, the performance of the developed methods is shown in Section 12.4. Finally, we draw the conclusion and discuss the future works in Section 12.5.

12.1 The Platform

KDE [1] is an advanced desktop platform which provides user-friendly graphic interface. It is an open source project. KBlocks is the Tetris game in KDE. We developed KBlocks to a platform for researches in artificial intelligence. The system components of KBlocks is shown in Figure 12.1.

[1] official cite: http://kde.org

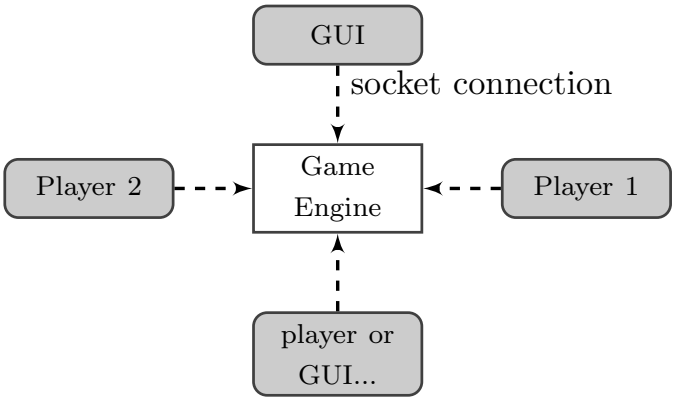


Fig. 12.1. The System Components of KBlocks

KBlocks can be run in two modes: KDE users can use it as a normal desktop game; researchers can choose to start a game engine, a GUI, or a player. The GUIs and the players are connected to the game engine via UDP sockets. The components can be run in one or several computers.

KBlocks can be configured with parameters defined in a text file. The game engine (and the GUI) supports game competitions among up to 8 players, in which one player could be a human. A hand-coded artificial player is integrated [72]. It can clear on average $2000 - 3000$ lines in single games. The competitions can be done in a synchronized mode, in which each player gets the new piece after the slowest player finishes the current placement.

A new artificial player can be integrated into the platform with ease. We provide a source code package in Internet [2], in which the class KBlocksDummyAI is a clean and simple interface for the further development. Graduate students can simply change the source code for their internship or thesis. Researchers can play around with some ideas or organize competitions.

Learning by imitation is developed for playing Tetris. The training data of the imitation learning are obtained from the imitated system. In this paper, they are the Tetris games played by the imitated player. We created several models to obtain the skills of the imitated player. The training process receives positive feedback if the models make the same decision as the imitated system. Otherwise, it receives the negative feedback. The imitation learning is successful if the trained models keep the similarity even if the data never appear in the training set.

[2] http://www.informatik.uni-freiburg.de/∼kiro/KBlocksDummyPlayer.tgz

The learning system consists of several components, as shown in Figure 12.2. We created three catalogs for these components: the data representation; the algorithms; and the learners. They are illustrated as the gray rectangles, the regular rectangles, and the round-cornered rectangles in the figure.

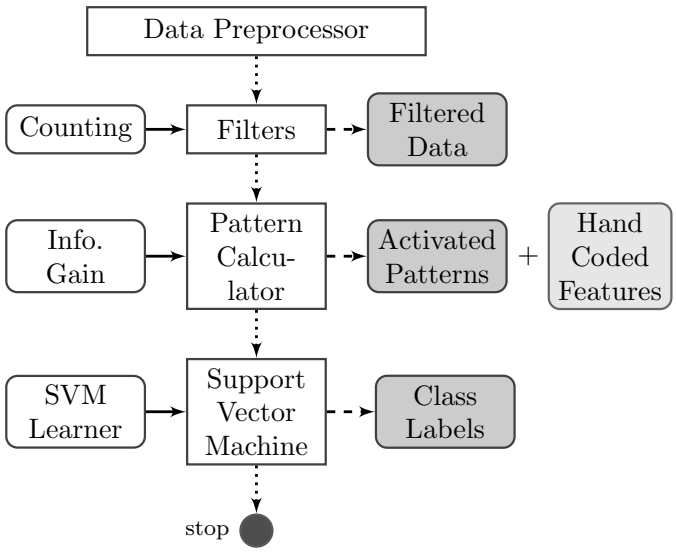


Fig. 12.2. System Components

Figure 12.2 also shows the relations among the components. We align these components vertically according to the catalogs. A lower algorithm uses the outputs of the upper one as its inputs. The learners computes the models which are used in the algorithms.

The middle column with the dotted arrow lines shows the sequence of the computation in the games. With the current board state and the piece (s, p), the data preprocessor can generate up to 34 candidate placements by enumerating all the rotation and the position of the p. The candidates are filtered because of the heavy computational power required by training the SVMs. The rests of the candidates are passed to the pattern calculator and the hand-coded features. Each candidate is transferred into a vector of the values of the patterns and the features. The vectors are used as the input of the SVM for the prediction. The output of the SVM can be described as how similar a candidate is to the choice of the imitated player. Consequently, the most similar one is labeled as the final choice.

12.2 Encoding Features

Training the SVMs is time consuming. There are 7 different pieces in Tetris: L, J, O, I, T, Z, and S. To place one of L, J, or T, there will be 34 candidates by combining all the possible rotations and positions; O has 9 combinations; I, Z, or S have 17. The candidate chosen by the imitated player is regarded as the positive case. The others are the negative cases. If the size of the training set is 10000, there are about 220000 tuples (cases) in the set. If each tuple is a vector of 39 values, training a SVM from these data would take more than a week using a $2.3GHz$ PC.

In order to reduce the data set, the types of pieces are used in the data preprocessor to separate the data into 7 subsets. Each subset is used to train its own filter, patterns, and SVM. In other words, seven SVMs work together in the artificial player.

When placing the current piece, human players can first reduce the candidates to a limited number by observing the surface of the accumulated blocks. Then, they choose one from the filtered candidates as their final decision. This idea was used to develop the filters for reducing the amount of data in the learning.

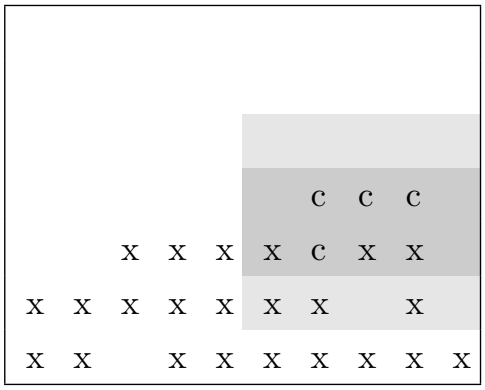


Fig. 12.3. An example of the Pattern

A filter consists of a set of patterns. Figure 12.3 shows the concept of the patterns. The current piece is denoted by 'c', it is an 'L' in the figure. We use 'x' to denote the already occupied cells. Around the placement, a small field, which is marked in gray, is chosen as the activated area for the patterns. The patterns are smaller than the small field. For example, the deeper gray area in

the figure shows a pattern. It contains 5×2 cells. The cells with a 'c' or 'x' inside are occupied.

A pattern can be activated by a placement. As mentioned above, the small field is activated by the placement. All the 5×2 patterns can be enumerated. We move a pattern around the small field. It is activated by the placement, if the occupied cells in the pattern match the occupied cells in the background (the small field).

Filters can thus be learned by counting. If a pattern is never activated by the placements of the imitated player, it can be used to reduce the candidates. Each filter is a set of such patterns. It can be learned by running the activation tests over all the training data.

12.3 Training Support Vector Machines

The patterns are useful not only in the filters but also for modeling the skills of the imitated players. For instance, a pattern was activated 1000 times over the training set, among which 900 were activated by the positive cases. This pattern cannot be used in a filter because there are mixed negative and positive cases. However, activating it apparently indicates that the placement tends to be positive because of the positive to negative rate in the training data. Therefore, the patterns are also used in this section to compute the inputs of the SVMs.

However, the patterns can only get the "local" information. They are checked within the small field around the placement. From another aspect, it is important to consider some "global" parameters in Tetris. For example, a candidate placement can clear 4 rows. This would be important for the game. The patterns, however, cannot express this occurrence.

We designed hand-coded features to acquire "global" information. If the patterns can define the tactics of the games, the features can be used to describe the strategies. In order to define these features, we use Figure 12.4 to illustrate some phrases: *hole, flat, column*, and *well*. A well or a hole is buried if it is no deeper than three cells from the surface.

The features are listed in Table 12.1. Items 2 and 3 are for the column. Items $4-6$ are about the flat. $9-11$ are for the hole. $14-18$ are about the well. Our features are compared with the features listed in [70]; the items with * were not mentioned. There are differences in the descriptions of the features because we use them as the inputs of the SVMs. The other researchers developed the evaluation function with the linear combinations of the weighted features.

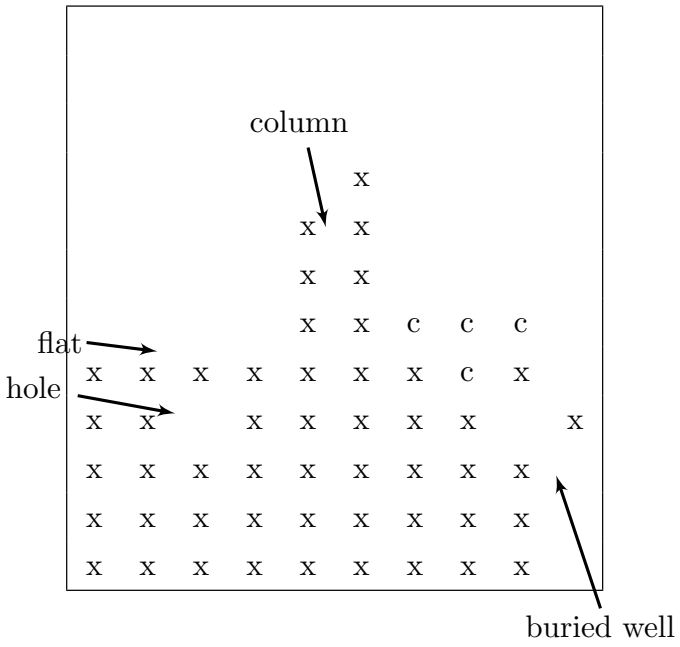


Fig. 12.4. The Illustrations of the Features

A large number of patterns can be created by enumeration. For example, an enumeration of 5×2 will create 1024 patterns. It is difficult to consider all these patterns as the inputs of the SVMs because of the required computational power. To our knowledge, there is no trivial way to compute a subset of the patterns which yield to the optimal performance of the SVMs. Therefore, we employ the information gain in decision tree for computing a subset of 20 patterns for each SVM.

SVMs are a popular method in data classification, in which the whole data set are globally classified with a set of the labels. Nevertheless, the data in Tetris are grouped by the current piece. Among the candidate placements of the current piece, the algorithm needs to choose the one which is closest to the choice of the imitated player. LIBSVM [20] provides an API to compute this probability, which is used in our implementation.

The values of the inputs should be within the same range in the SVMs. The patterns always have a value of 0 or 1, which denotes whether or not it is activated by the current placement. The value of features, however, can be much bigger. For example, the maximum length of the flat can be up to 9

Table 12.1. List of Hand-Coded Features

1*	How many attacks are possible after the current placement.
2	The number of the columns.
3	The increased height of the column.
4	The increased number of the flat.
5	The decreased number of the flat.
6	The maximum length of the flat
7	The increased height of accumulated blocks.
8	The height difference between the current placement and the highest position of the accumulated blocks.
9	How many holes will be created after the current placement
10	How many holes will be removed after the current placement.
11*	How may occupied cells are added over a hole.
12	The number of removed lines of the current placement.
13*	How well will the next piece be accommodated.
14	If a well is closed by the current placement, how deep is the well.
15	If a well is open by the current placement, how deep is the well.
16*	How may occupied cells are added over a buried well.
17	The number of the open wells.
18	How deep is the well, if it is created by the current placement.
19	Whether a well is removed by current placement.

in a standard Tetris game. In order to avoid this situation, the values of the features were mapped to 0, 0.5, or 1 in our implementation.

12.4 The Experiments

The experiments were done in a grid system. There are 8 computers in the grid. Each computer has 8 $2.3GHz$ AMD CPUs, and $32G$ memory. 64 processes can be run in parallel in the grid.

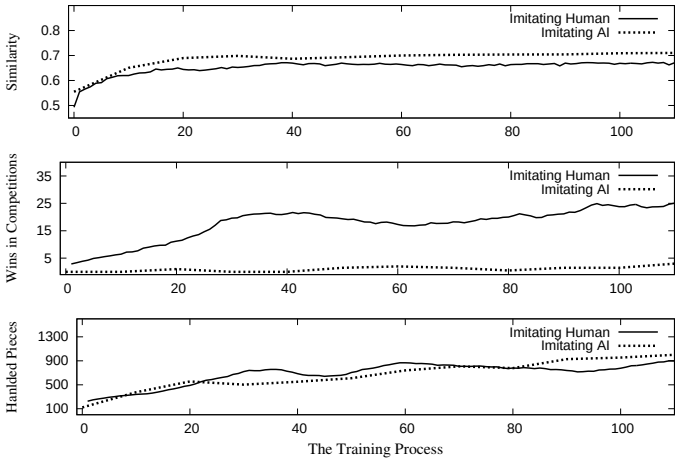


Fig. 12.5. The Training Process

We recorded 10 games of a human player. Each game lasted more than one hour. The game speed was limited, so that the player had enough time for the game. The player can play Tetris at an amateur level. In total 6720 rows were cleared in these games. The human player was regarded as the first imitated player.

The Fehey's artificial player [31] was run for about 1 hour. It cleared 6774 lines without a restart. The game was recorded as the training set. Fehey's artificial player was the second imitated player.

The two imitated players had very different behaviors in the games. If the human player competes with the artificial player in the synchronized mode, the artificial player has very little chance to win, because it attacks only a few times.

The recorded data were divided into 150 subsets, 120 of them were used as the training set. The rests comprised the testing set, through which the similarity between the trained models and the imitated players can be calculated the rate that the trained model chooses the same placements as the imitated player. The results are shown in the upper plot of Figure 12.5. The data were averaged over 10 slices.

The solid lines show the performance of the player that imitates the human player. The dotted lines are the player that was imitating Fehey's player. Both imitations achieved a similarity of about 0.7. The curves resemble a typical learning curve because the similarity is regarded as the evaluation in the learning. The similarity cannot be higher because of the differences in

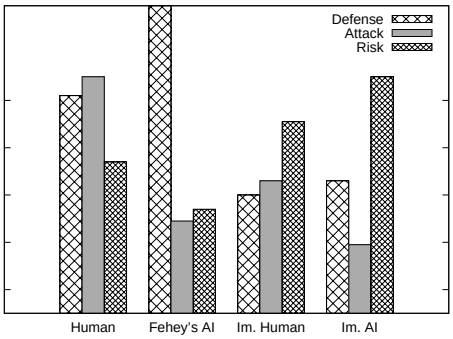


Fig. 12.6. Behavoiurs of Different Players

the data representation and the models between the imitating system and the imitated systems.

The trained models compete against Fehey's player in the synchronized two-player games. 200 random piece sequences were generated for the 200 games, so that each model was evaluated in the same set of the games. The middle plot in Figure 12.5 shows the winning rates of the imitating players. The player imitating human finally achieved 0.25 as its rate of wins in the competitions against Fehey's player. The other imitator did not perform well because the competitions were between the imitating and imitated systems. As the similarity cannot be very high in our implementation, the imitated system should in principle be better than the imitating system.

The trained models also play the single games. The piece sequences used in the games were generated and fixed. The number of handled pieces was used as the evaluation of the player. The results are shown in the lower plot in Figure 12.5. Fehey's artificial player is better than the human player in the single games, which explains the observation that the imitator of Fehey's player is in the end better than the other imitator.

The training process was designed to search for the maximum rate of the similarity. The rate reached 0.68 at the 30^{th} data slice, and kept this value after that. The performance in the competitions and single games can still be improved after the 30^{th} data slice. This observation indicates that a bigger training set helps to improve the game skills, though it does not improve the similarity in the imitation.

The human player, Fehey's player, and their imitators have different behaviors in the games. In order to show the difference, we designed the evaluations for the attack, defense, and risk. Each player played the same sequences of the pieces in the single games. Attack is the average number of attacks that

the player made to clear 100 lines. Defense is evaluated by the average number of cleared lines of each game. Risk is measured by the average height of the placements. The results are shown in Figure 12.6.

Fehey's player has a defense ability several levels of significance better than the other players. This information was shown as the open-end column in the figure. The other evaluations were mapped to a comparable range. The human player has the best attack ability, which explains how its imitator has chances to defeat Fehey's player in the competitions. The two imitators show quite different behaviors according to the evaluations, which means our imitation learning can generate various artificial players according to the imitated systems.

12.5 Discussion

In this chapter, we developed a platform for Tetris competitions. The platform is based on an open-source project. The GUIs and players can connect with the game engine via the socket connections. A dummy player was provided as an interface for further development.

We implemented a framework by using learning by imitation. The framework consists of several sets of filters, pattern calculators, and SVMs. The imitation tasks were mapped to a standard data classification problem. The experiments show that our imitators have chances to defeat Fehey's player, which is the best-known artificial player in single Tetris games. And the imitation learning can acquire diverse skills in Tetris games.

This artificial player is implemented according to the SAL framework. There are multiple learners in the framework. The learned player can be used to select an interesting game for further training. The inputs and outputs of the learners form a loop so that each performance of one of the learners create improvement space for the incremental learning.

The imitator did not win many games in the competitions. In the next step, we will develop an extra learner for the better results. The initial experiments showed that the wins in the competitions can be significantly improved by using the rate of wins as the evaluation in the learning.

So far, we developed two players for Tetris. The first one was developed in Chapter 11, which is based on bandit based Monte-Carlo planning. It requires to foresee ten pieces in advance for the planning. The second one is not as competitive as the first but it requires only one piece in the future and it consumes much less computational power.

Part IV

Other Applications

13

A Game Controller Based on Multiple Sensors

There are many fantastic digital games. However, playing such a game is regarded as an unhealthy activity [24]. The main reason is that most of the games can be fully controlled by hand. It would be unhealthy if children stayed on the couch for several hours, gazing at the screen and moving only their fingers.

"More Movement" is a popular idea for playing digital games. This concept can be found in current commercial-available game consoles. For example, accelerometers are integrated into the remote controller of *wii*[1], which requires players to twist their wrists. A camera (*EyeToy* and [2] *Natal* [3]) is used to recognize the gestures of a player, so that one needs to wave their arms to control a game. Although these devices introduce "more movements" than a normal joystick does, their effects are still limited.

13.1 Motivation

The main idea of the work here is to enable the players to control the game by their activities, which involve not only hands and feet, but also the movements of the whole body. In addition, there could be other inputs such as the voice. Different from the idea of a single device, e.g. joystick, the proposed game controller can have multiple parts (sensors) scattered at different places in the environment. The controller can be used not only for a single player, but also for a team of players cooperating on a task. It raises the question on how to configure the mapping, which is from real-world activities to a sequence of

[1] wii is a product of Nintendo
[2] EyeToy is a product of Sony
[3] Natal is the future concept for the game console of Microsoft

commands to control a game. Some algorithms in artificial intelligence, such as finite state machine, can be employed in this mapping.

This chapter is structured as follows. In the next chapter, we discuss the relations between this work and the literature. Then the implementation of the game controller is depicted. After that, two different mapping algorithms are implemented for a fighting game. The experiments show the quality of the control and people's attitudes towards the system. In the last chapter, we conclude and present the outlook for future work.

13.2 Related Work

Ubiquitous computing, first proposed by Mark Weiser in 1988 [83], is receiving more and more attention in the domain of Human Computer Interactions(HCI). The main reason is that there are many embedded systems coming into our daily lives, which foster new ideas in HCI. Many works in HCI can be described as the replacement of the two interfaces: one is from human to computer, which are a mouse and a keyboard; another is from computer to human, which is a display. Instead, physical interfaces are developed, which are supposed to work in a way that the computer in a system is totally hidden behind these interfaces [36]. The end users of such a system cannot aware that a computer is being operated.

In some applications, the computers need to be directly observed and operated. The subject of this paper – digital games – can be classified into this kind of application. The reason is that the display of a digital game, e.g. the game of a simulated war, is actually a highlight of the system. For this kind of application, an important idea in the direction of HCI was proposed by Ishii and Ullmer in 1997: the operations of the virtual objects in the digital world are achieved in a "touchable" way, which is the so-called *Tangible Bits* [41]. To develop a game controller, researchers normally focus on how to create a physical interface to control the game, while the display is left untouched. Headon and Curwen developed a ubiquitous game controller in 2002 [39]. Sensors are installed under the floor so that the player's movements on the floor can be transferred to control a digital game.

Varieties of devices (sensors) have been used in game controllers in the past number of years. For example, RFID-System can be used as an input of a game [12], as well as voice [38]. The VoodooIO gaming kit [75] consists of configurable press-buttons. In this work, there are multiple sensors working together in the environment, which is the idea of ubiquitous computing [83]. Our game controller has a similar distributed architecture to [39] and [75]. A

set of devices are integrated into a single controller, which is very different from the ones employed only one input method [12, 38]. LMS is used for observing the surroundings. To our knowledge, the approach in this paper has three highlights: the game controller is based on a series of advance technologies which are real-time and require no physical contact for the control; the mapping module in the framework can significantly change a game, which makes a game, rather than easy and comfortable, exhausting for the players; the experiments show statistics on people's attitudes towards the mapping and the developed game controller.

13.3 Game Controller

13.3.1 System Architecture

The system architecture is shown in Figure 13.1. There are three modules in the system: sensor module, mapping module, and game control module. The sensor module are distributed in several computers, which provides a flexible way to place the sensors at the proper places and to add computers for more computational power. A local area network (Ethernet) is established for the communication. The communication between two processes is achieved via a "socket". It takes less than $10ms$ on sending and receiving a message.

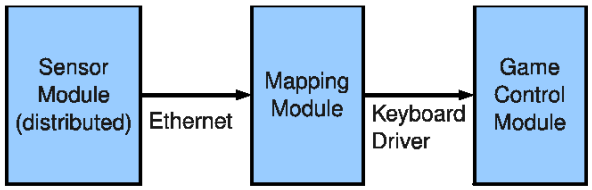


Fig. 13.1. System Architecture

The mapping module is a must for the system. Most commercial games assume that they are controlled by pressing buttons (and) or a mouse, which

should be done with ease by hand. In many cases, the skill of playing a game is to press the right buttons at the right times. When the game is controlled by an activity, the tasks are totally changed. Jumping and running from one place to another need time spans and efforts which are very different from pressing buttons. A game, designed for hands, can hardly be controlled by an activity. The main functionality of the *mapping module* is to bridge the gap between using "hands" and using an "activity" to control. It also provides a way to configure the game behaviors. The arrow between the *mapping* and the *game control* modules is implemented by hooking the keyboard events in the operating system.

All the components of the controller can be put into a mid-size suitcase, although the valid acting area of the controller can be as big as a room. The controller does not have many requirements of the environments. For example, it does not require a critical lighting condition as a normal camera does. It can be setup in short in a any typical room, or even outside. One shortcoming of the system is that its price is not yet ready for market.

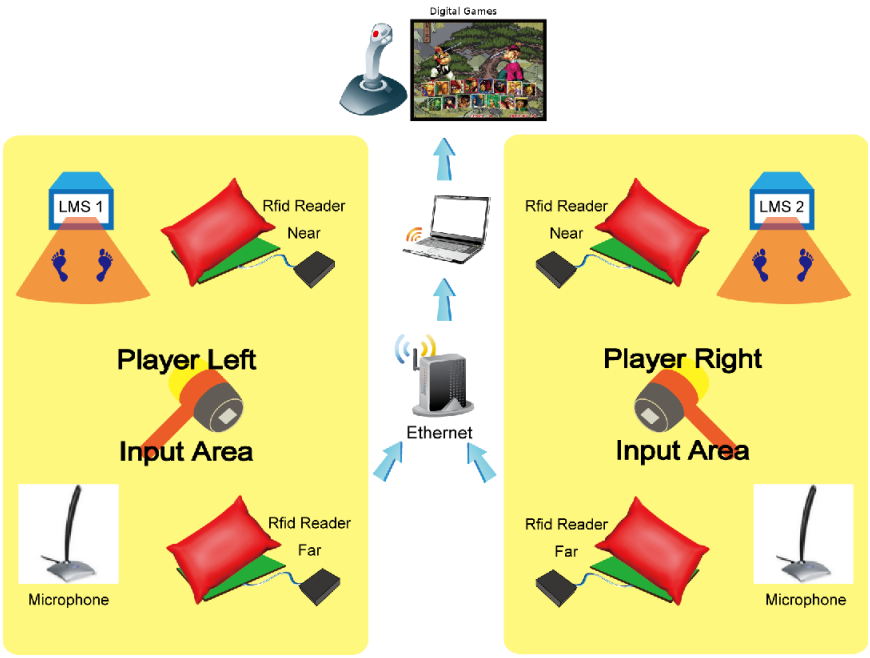


Fig. 13.2. System Components

Figure 13.2 shows the system components. For each player, we use one LMS, one microphone, and two RFID-readers which can drive the player to walk or to run in between. Two sets of the devices make it possible for a human versus human game.

13.3.2 Observation of the Legs

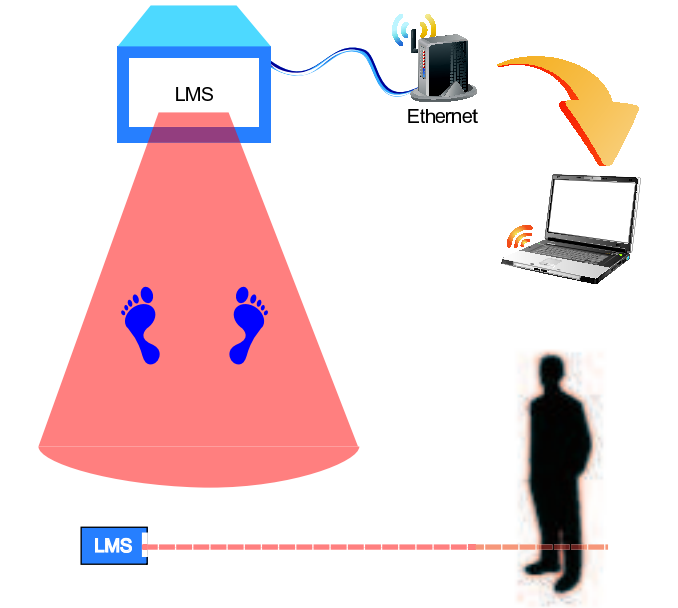


Fig. 13.3. Observing the Actions of the Legs Using LMS

Laser Measurement System (LMS) is a fast and accurate device to observe the surroundings. It emits a fan-shape laser beam. The measurements

are a series of points, each with a distance and angle to the laser start point. *LMS400*[4] is chosen to observe the legs' actions. It has a scan-frequency of about $350Hz$; the laser open is $70°$; the valid range of the measurements is from $700mm$ to $3000mm$ with an accuracy of $\pm 3mm$. The LMS is set up on the ground in such a way that the laser beam projects on the lower part of the calf, as shown in Figure 13.3. When the player stands still, LMS can observe two round objects in its view.

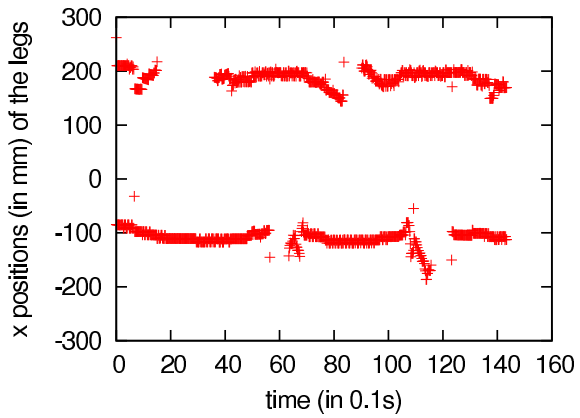


Fig. 13.4. The Plot of the Observed Actions

In order to detect the legs, an algorithm is implemented to trace the position of each leg. Figure 13.3.2 shows these data: x axis is the time, y axis is the x position of the legs; the observed player was walking. One leg is raised after another when a person is walking, and the LMS cannot observe a leg if it raised higher than the laser beam. We can see in the data that the player walked for four steps. The sensor data is noisy, thus the recognition is implemented using a moving window.

13.3.3 RFID-Tags and Voice as the Input

Radio Frequency Identification (RFID) is a technology by which an ID can be read without any physical touch. It can identify many IDs. If the IDs are combined with other objects, e.g. the left hand and the right hand, the RFID system can recognize much more information than a press button. We chose

[4] LMS-400 is a product of SICK AG

an RFID reader and tags with a reading frequency of more than $100Hz$, and a range of $150mm$. Two RFID readers are used for each player, who has two toys with the tags. The readers are placed at the two ends of the active area, so that reading a tag at one reader and then the same tag at the other is an activity of walking (running) through the area.

Recognizing the content of a voice is a difficult task. To simplify the task, we use only the volume of the voice to control a game. If the output of the microphone is only zero (the background noises) and one (shouting at the microphone), a single threshold can lead to accurate and robust performance. As an input, a shout only needs to be loud enough. It also adds funny effects. Shouting during a game causes a lot of laughter for both the players and the spectators.

13.4 Mapping Methods

We chose a fighting game (*Samurai II*[5]) for several reasons. It needs a complex control sequence. Board games such as "FreeCell" and "Chess" are much less challenging. In addition, human versus human mode makes the game more exciting. The tasks of the game are quite simple: attacking the opponent as much as possible, while defending against being attacked. The character is controlled by three direction keys, and three action keys. In addition, each character has three or four advance actions, e.g. a sequence of right, down, left, boxing, and jumping will lead to a rush slash. The advance actions require more skills in the control, and in most cases they decide the results of a game.

In our system, the game is controlled in three distinguished modes. "Joystick" is the classic way to control a fighting game. This mode is regarded as a baseline for the estimation of the others. "Direct mapping" is an intuitive and natural way to map the activities of the player to the actions of the virtual character in the game. "Indirect mapping" mode mentions the activities of the player, in which the game is significantly simplified.

In the "direct mapping" mode, the human actions are directly mapped to a game action. The mapping is designed as intuitive and easy as possible. For example, the jump of the player is mapped to the jump of a virtual character. This is even more intuitive than "pressing a jump button". We also choose the toys similar to the weapons in the game. RFID tags are attached on an inflatable hammer and a rubber sword. Although the player should be much more active than when they are in the "joystick" mode, the "direct mapping" mode

[5] Samurai II is a product of Neo Geo

keeps most highlights of the game. Most basic and advance actions available in the game are still possible by using the game controller. The mapping of the advance actions is designed to be as close to the real movement as possible. For instance, if the player inputs using a sword, and then jumps, the game character will do the most similar action: a rising slash. The main disadvantage of the "direct mapping" is that it delays a bit from the time the player starts the action sequence to when the virtual character reacts to the command, especially when there is a long sequence of actions that need to be performed.

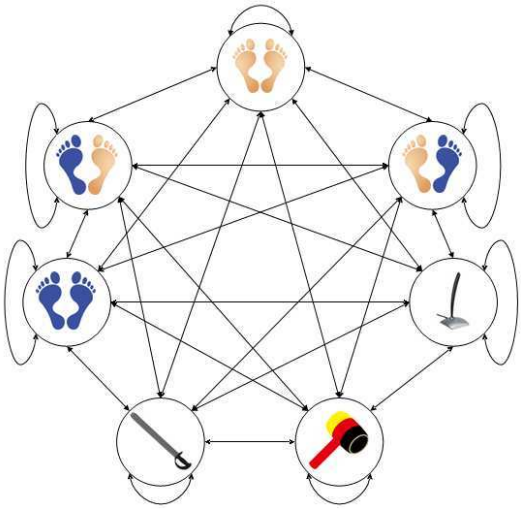


Fig. 13.5. A Full-Connected Finite State Machine

In "indirect mapping" mode, we use a finite state machine to control the game. Figure 13.5 shows an example of such a state machine. Each node in the graph is the input of a device. The activities of the player can be mapped to a sequence of states and transitions. The activities of a player can thus be modeled and expressed in an *indirect* way. Considering the chosen fighting game, we configured the state machine as follows. Each transition is weighted by the required energy from one node to another. The weights are accumulated in the state machine – for example, the jump of a player can be mapped

to a sequence of observations: two legs are observed, no leg is observed, and two legs are observed. The sequence is scored as 10. Walking activity can be recognized in a similar way, which is scored as 5. The accumulated energy can be released to the virtual character of the game by a special node, e.g. sword. The "indirect mapping" significantly changes the game. The only skill a player needs is to release the energy at a proper time, which makes the game very easy for the beginners. The player would feel quite exhausted after only two or three rounds.

13.5 Experiments

The experiments were done in the following scenario. First, the participants play the game with a joystick, so that they get to know the basic operations and the advance skills of the game. Then, they are invited to play against each other using the "direct mapping" mode. Before they start to play, we give a short introduction to the mapping. The instructions are introduced and demonstrated one by one. Then, they can play by themselves while looking at the instruction page. After that, the "indirect mapping" mode is introduced in brief and they are asked to play using this mode. Finally, they evaluate and compare three control modes by completing a questionnaire.

The evaluation takes the form of scores: minus is negative; zero is neutral, and plus is positive. Questions $1-4$ are used to evaluate the quality of the control. They cover four aspects: easy to use, reactive, accurate, and fully-controlled. Question 5 is designed to evaluate whether or not the game controller is novel for the participants. Question 6 estimates how healthy a digital game could be. Question 7 ask the participants to rate the three control modes overall. Finally, we got 31 participants, 28 of which completed the questionnaire.

Figure 13.6 shows the sum of the scores of the questions $1-4$. The joystick has a better control quality over all of the four aspects. There are mainly two reasons for this result. On the one hand, a joystick is a classic control device, thus most people are good at using it. On the other hand, the game was originally designed for running in a "game machine", which has a control interface very similar to a joystick. The "direct mapping" turns out to be more difficult, less reactive, and less accurate to be controlled than the "indirect mapping" in the experiments. "Direct mapping" is more difficult because the mapping contains an instruction list, which is new for all the participants; they have to learn these instructions during the games, whereas, the "indirect mapping" has an extra GUI and the state machine calculated whatever the

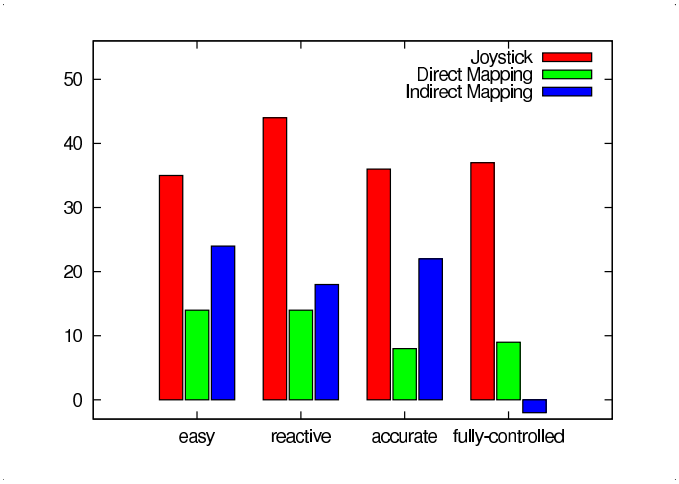


Fig. 13.6. The Quality of the Control

player did as a certain amount of energy, which is quite reactive. The final *trigger to release* action is very easy. Therefore, the control using "indirect mapping" can be more reactive and accurate. "Indirect mapping" was evaluated as negative in the *fully-controlled* because the game actually cannot be fully-controlled in this mode.

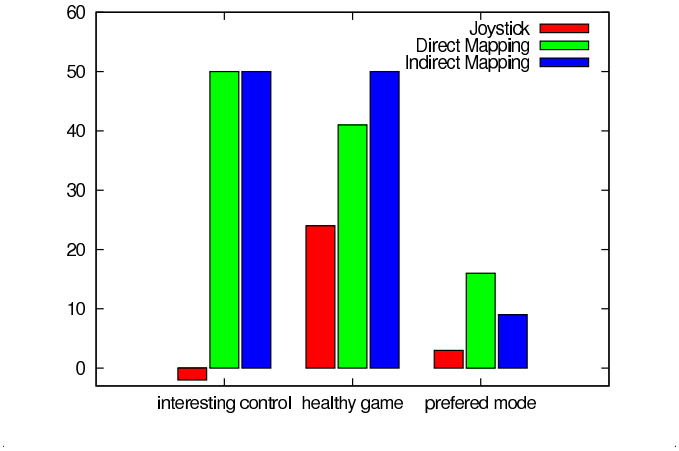


Fig. 13.7. The Attitude towards the Control

Figure 13.7 shows the attitudes of the participants towards the three control modes. The joystick was not interesting for the participants. The other two modes were equally novel. Many people gave surprisingly positive feedback towards the joystick on *healthy game* question. A reasonable explanation is that the evaluation process was quite short in time; participants were not aware that moving only their fingers and gazing at the screen for long periods of time is somehow unhealthy. Compared to the other modes, "indirect mapping" induces movement the most, so it was estimated as the most healthy game. "Direct mapping" was also significantly better than the joystick. The participants were asked to exclusively choose a favorite mode in the last question. 16 people preferred the "direct mapping", even if the control quality of the mode is the worst one. 9 people preferred "indirect mapping"; only 3 people remained with the joystick. A possible explanation is that "direct mapping" not only introduces much more movements, but also keeps the most interesting feature of the original game. "Indirect mapping" is also interesting, but it requires much less skill.

13.6 Discussion

In this chapter, A game controller is developed by using three distinctive measurement methods, which require no physical touch. We chose a fighting game as a case study. Two different mapping modes were implemented. People participated in the evaluation of the game system. The experiment's statistics show that *mapping module* can significantly change the game. The game controller was pretty interesting for the participants. It achieved a fairly good evaluation in the quality of the control. The participants preferred to use the controller, although it did not have a control quality as high as that of a joystick.

Instead of making the game control easier and more comfortable, we are working towards making the control be taxing or even exhausting for the players. The experiments show that people actually accept this idea quite well. Most of the participants preferred to play the game using the developed game controller, rather than a classic joystick. It is possible that some highlights of a game are limited by the controller, which is a shortcoming of the system. The mapping method needs to be chosen very carefully.

In the future, other measurement techniques can be integrated into the controller. For example, if there is an *eye tracking* camera by which the computer knows where the player is looking, the operations of a mouse become feasible for the game controller. The activities of players could be studied. In

this work, we did not actually pay much attention to this. It would be wonderful if an interesting sport could be combined with a digital game.

14

On-Line Detection of Rule Violations in Table Soccer

14.1 Introduction

Table soccer also known as Foosball is a popular game, often played in pubs or other social contexts. The ball can be kicked by rotating the rods, and professional players can shoot the ball across the table within a few hundred milliseconds. Because of the high speed of table soccer, a human is not always able to thoroughly observe all actions in the game, such as the fast turning of a kicking rod. However, this is necessary to make objective decisions about rule violations.

We present an approach to automatically detect rule violations on-line, using high frequency sensor data to classify and evaluate game situations. This helps humans playing table soccer. Most rules covering game mechanics depend on the detection of a playing figure kicking the ball. Thus, detecting violations of one of the most important rules is the main task: Rods may not be rotated by more than 360^o before or after ball contact. This prevents players from spinning the rods in an uncontrolled manner when kicking.

Because the sensor data is noisy, a probabilistic model for kick detection is required. Supervised learning is used to train a naive Bayes classier for this purpose.

The chapter is organized as follows. First, the related work is addressed. Then, the methods used for segmentation and classication are explained in Section 14.3 and 14.4, followed by the experimental setup and results in Section 14.5. Finally, we will conclude our work in Section 14.6.

14.2 Related Work

This chapter is based on the kicker recorder, KiRe [88] and the ball detection method of the KiRe [25].

A common approach to model dependencies between random variables is a *Bayesian network* in the form of a directed graph [57]. Nodes represent random variables or classes, while directed edges denote dependencies. In a classication problem, the network is used to compute the posterior probabilities of all classes c_k given the observed values of the attributes: $P(c_k|x_1, ..., x_n)$. The class with the highest probability is then assigned to the observation. Using Bayes' Theorem, the posterior of a class c can be computed as Equation 14.1 which flips the conditioning to make the distribution easily learnable.

$$P(c_k|x_1, ..., x_n) = \frac{P(x_1, ..., x_n|c)P(c)}{p(x_1, ..., x_n)} \tag{14.1}$$

With the assumption that all classes are equally probable, all attributes independent given the class and that the prior is independent of the class. This greatly simplifies computation to Equation 14.2. With being a normalization constant, this is the naive Bayes classier. Even though the naive assumptions seldom hold in reality, naive Bayes has been attested a good performance in many domains and can even be the optimal classier with respect to the misclassication rate in some cases [28] [90].

$$P(c_k|x_1, ..., x_n) = \alpha \Pi_{i=1}^{n} P(x_i|c) \tag{14.2}$$

In this work, observed values are not discrete but continuous. One common approach is to discretize the attributes [29]. However, we believe that discretization will lead to system degradation by discarding information like the probability for a kick. Instead, we model the random variables to be Gaussian distributed and use the joint probability of the distributions for each variable.

Common extensions to naive Bayes include tree-augmented naive Bayesian networks [35], where additional dependencies between the attributes can be modeled. The cost is a higher algorithmic complexity. To cope with the high sampling frequency to observe games on-line, an ecient implementation is needed in our case. Our experiments revealed that the classication performance of naive Bayes is good enough for this application.

With continuous and Gaussian-distributed attributes in Equation 14.2, it follows the equation 14.3. Variance δ_i^2 and mean μ_i of each attribute i are determined with supervised learning for a known class label c. m attribute

values $x_{i,1}, ..., x_{i,m}$ are sampled for each attribute i and the unbiased estimators for mean and variance are used on these values.

$$P(c_k|x_1, ..., x_n) = \alpha \Pi_{i=1}^{n} \varphi(x_i) = \alpha \Pi_{i=1}^{n} \frac{1}{\sqrt{2\pi\delta^2}} e^{\frac{-(x_i-\mu_i)^2}{2\delta_i^2}} \tag{14.3}$$

14.3 Segmentation of the Data

To cope with the high speed of the game, the sensor data is read and recorded at roughly $200Hz$ using a standard PC, distributing the data stream over network for further evaluation on other machines. The ball is located with a Sick LMS400 laser measurement system, scanning through the gap between table surface and feet of the playing gures. Rod positions are measured with optical distance sensors and rod angles are observed with magnetic rotary encoders. Overall, there is almost no additional friction on the rods, enabling an unhindered game play.

To analyze the game state, one needs to segment the sequential sensor data and detect key events in it. The sensors introduce Gaussian-distributed noise on the measured signal. Currently, three rods are observed by sensors, all in the same half of the table: blue attacker, red defender and red goalkeeper.

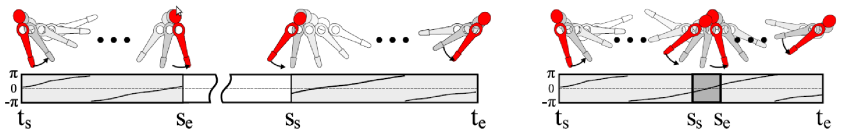


Fig. 14.1. The Spin of a Game Rod

With respect to the rule violations described in Section 14.1, a rod spin is completely parametrized by four timestamps: t_s and t_e, the starting and end times of the whole spin, as well as s_e and s_s, the times when the first 2π part of the rotation ends and the time when the last 2π part starts, shown as in Figure 14.1. These discrete timestamps need to be detected in the stream of angle measurements for each rod i.

$$\Delta_i(t) = \alpha_i(t) - \alpha_i(t-1) \tag{14.4}$$

To do so, the dierence in between two successive rod angles α_i is observed as approximation of the derivative, shown in Equation 14.4. When crossing from 2π to 0, the values are adjusted accordingly. As soon as $\Delta_i(t) \geq \epsilon$ for some small threshold ϵ, the start of a spin movement is detected, and it lasts until $|\Delta_i(t)| < \epsilon$. To reduce the inuence of noise, the signal is smoothed by using the running average over a window of size three.

As soon as $\alpha_i(t_s)$ is passed for the second time, the time s_e is detected. s_s depends on $\alpha_i(t_e)$, the angle at which the spin stops, and is detected by using a circular array or ring buffer indexed by angle. For additional robustness against noise, monotonicity in between start and end of the spin is enforced when storing timestamps in the circular array. Also for noise robustness, all bins in between storing two successive timestamps need to be emptied.

14.4 Kick Detection

Naive Bayes classiers are used to detect kicks, by using the relation of ball and active gure as input. The active gure is the one closest to the ball on the rod that is within range of the ball. The two directions of a kick are distinguished by classifying two cases, forward and backward.

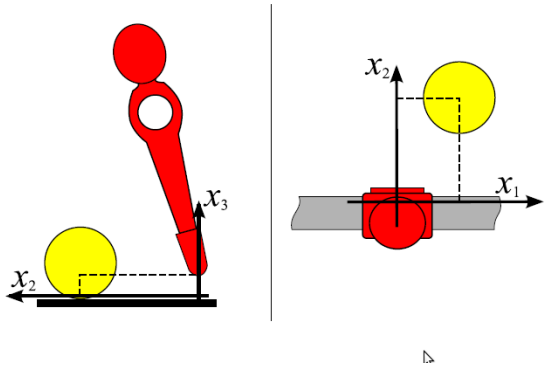


Fig. 14.2. The Parameters of the Kick

Input for each classier are the continuous attributes x_1, x_2, x_3, computed from coordinates of ball and active rod, and its angle. Figure 14.2 displays the coordinates relative to the playing gure, Figure 14.3 displays the resulting

Bayesian network. Note that x_3 is directly related to the vertical distance between ball and playing gure, because it is assumed that the ball always touches the table when kicking. When it is not on the surface, e.g. resulting from a fast kick, it cannot be observed by the laser anyways.

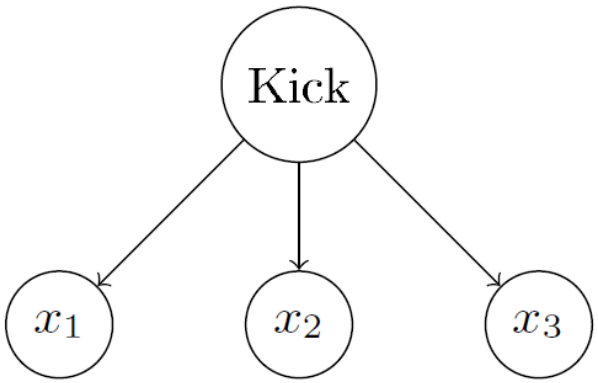


Fig. 14.3. The Naive Bayes Classifier of the Kick Action

The probability can now be computed according to Equation 14.3 with $n = 3$ variables. In the implementation the log-likelihood is used instead, turning the products into sums. This is computationally more stable because very small probabilities are avoided, while it is equivalent in terms of classication. Furthermore, the constant is ignored for classication because it is assumed to be identical for all classes.

To detect violations of the rod-spinning rule eciently, the log-likelihood for kicks is constantly recorded. As soon as a spin covering an angle of more than 2 is detected, the peak of the kick likelihood in the intervals $[s_e, t_e]$ and $[t_s, s_s]$ (see Figure 14.1) is compared to an experimental threshold. Depending on the direction of the spin, the forward or backward probability is used.

14.5 Experiments

In the experiments, we rst evaluate the supervised learning performance of the kick classier. Then, the parameters for the kick model for on-line detection are

Table 14.1. Parameters of Training by Sampling

i=	1	2	3
Mean μ_i	-1.37	23.54	13.02
Variance δ_i^2	30.30	7.09	3.21

trained. Finally, we test the performance of violation detection in real table soccer games on-line.

To evaluate the learning performance, 50 recorded kick actions were randomly partitioned into a test set of size 10 and a training set of size 40. In each single action k, the instant when the playing gure touches the ball is selected by hand, and the variable state $(x_{1,k}, (x_{2,k}, (x_{3,k})$ is extracted. The parameters of the kick model mu_i, δ_i for $i \in \{1, 2, 3\}$ are then learned incrementally, using $k = 1, ..., 40$ samples of the training set as input. The performance on classifying the test set is evaluated for each step.

$$V(k) = \Sigma_{l=1}^{10} \gamma_k \Pi_{i=1}^{3} \varphi_{i,k}(x_{i,l}) \tag{14.5}$$

As performance measure V, the sum over the normalized probability of each test sample $x_{i,l}$ is used, shown as in Equation 14.5. The Gaussian distribution $varphi_{i,k}$ is parametrized by mean and variance of variable x_i, using k samples of the training set. The normalization constant is determined by the maximum of the joint probability, reached at the mean, as shown in Equation 14.6. This scales all Gaussian distributions in Equation 14.5 to the range of $[0, 1]$. Otherwise, their probabilities would not be comparable, because a more general model creates a more shallow distribution.

$$\gamma_k = \Pi_{i=1}^{3} (\varphi_{i,k}(\mu_{i,k}))^{-1} \tag{14.6}$$

The resulting learning curve is shown in Figure 14.4. A first peak is reached after using only six samples for training, and the performance of the learned model stays stable after using 16 samples for training. A training set signicantly larger than 20 samples only leads to small improvements.

To classify kicks in running games, only the model for a forward kick is learned. The parameters for a back kick are obtained by negating μ_2. To train the parameters of the kick model, the variable state (x_1, x_2, x_3) is sampled from several congurations with the ball in front of a playing gure, varying positions, angles and playing figures.

Table 14.1 displays the parameters trained through the sampling process, using 27 samples. As expected, the peak is centered in front of the playing

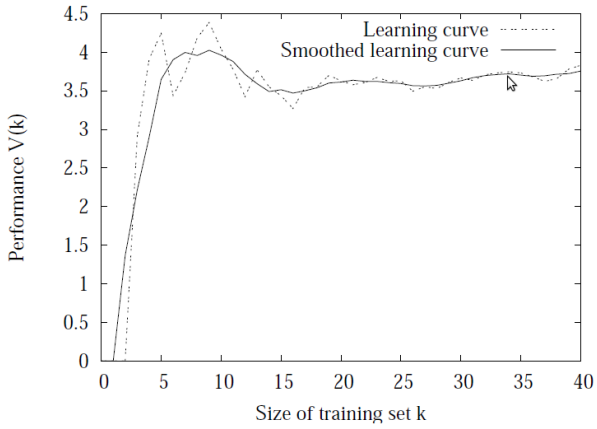


Fig. 14.4. The Learning of the Kick Action

gure. The rectangular footprint of the playing gure results in a larger variance in x_1 than in x_2.

Spins covering angles of more than 2π and the classifying timestamps t_s, s_e, s_s, and t_e are successfully detected. Figure 14.5 exemplarily shows the sensor recording of one spin movement starting at $t_s = 31$ and ending at $t_e = 105$, with the detected timestamps marked. Note the noisy signal for $\alpha(t)$, and that the whole spin from the angle $\alpha(t_s) \approx 5.85$ to $\alpha(t_e) \approx 4.75$ (rad) lasts only 74 sensor ticks, which is about $0.3s$.

To evaluate the detection rate in real games, test subjects not familiar with the system played several games on the table. The position of the players on the table (attacker or defender for the blue or red team) were exchanged regularly. Two players rated themselves as good, two average and the remaining seven as amateurs. In total, 9 games were recorded for about 105 minutes, silently logging the detected rule violations. Before the games, the rod spinning rule was explained to the players, and they were asked to evaluate whether they think they violated the rule after each shot. Afterwards, the recorded sensor logs were manually inspected in slow-motion for rule violations, and compared to the detections of the system and the players.

All in all, there were 42 rod spins of more than 360 degrees on the observed three rods, of which 19 were illegal kicks. The players themselves were aware of only two of them (10.52%), while the system detected 17 violations correctly (89.47%). Two violations were missed by the system, and one false positive detected. When a rod is spun and misses the ball closely

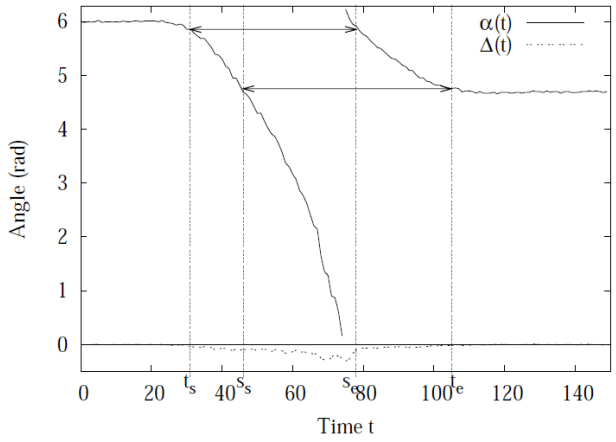


Fig. 14.5. An Example of Angles with a Spin

instead of kicking it, there is a chance that a rod spin violation is detected. The noise on the rod and ball data might add up so that the ball is falsely located as being touched by the rotating gure, resulting in the false positive detection. False negatives occur for similar reasons.

The referee system runs eciently on a standard PC with a 2.66 GHz Pentium 4 CPU and $1GB$ RAM, running SuSE Linux 10.1. The application uses just a small fraction of the available CPU power. Most of the CPU power is used for a $3D$ display of the soccer table in the user interface, which shows the live representation of the soccer table and slow-motion replays of rule violations. The distributed implementation of the system allows the display to be easily outsourced to a dedicated machine, leaving more processing power for example to detect additional rules.

14.6 Discussion

An approach is developed to automatically detect rule violations in table soccer games. A naive Bayes classier is trained off-line to detect kicks, using the relation of ball and closest playing gure as input.

The classier demonstrated a good performance in the on-line classication experiments, detecting 89.47% of all rule violations, while the human players only detected 10.52%. The ecient implementation of naive Bayes enables the system to run eortless on a standard PC, evaluating the high-frequency sensor

data on-line during a running game. All this demonstrates the usefulness of our implementation. Additional robustness on the classication method can be achieved in future work by taking the ball movement into account for kick detection.

Finally, future research on table soccer can benet from the classication method described here, such as learning by imitation or game analysis. The classier could be used to detect various relations between ball and playing gures.

Part V

Conclusion and Discussion

15

Conclusion

Switching Attention Learning (SAL) is a learning paradigm on which several learners can be developed and connected. The connection among the learners refers to the input of a learner comes from the output of the others. In SAL, these connections should form a loop. In boosting, co-training, and multi-view learning, a similar loop structure can be found. One of our innovations is that SAL has a more general definition, making it applicable to a wider range of applications.

SAL is applicable in a scenario in which an agent can observe and imitate other intelligent systems. These systems could be human beings, or other agents in computers. Game competition is a suitable test-bed for SAL because the agent developed using SAL can observe, imitate, and compete against its opponents. In the scenario, the agent should have similar control abilities to its opponents. Two games, table soccer and Tetris, were chosen as the test-beds for SAL.

The development of the first test-bed – table soccer – is based on the existing robots: KiRo and Star-Kick. Both of these robots can control one side of the game, competing with human players on the other side. However, KiRo and Star-Kick cannot observe the actions of the human players, thus it cannot properly react to them. To develop the robots further, the extra sensors should be mounted on the robots for the observation.

To observe the behaviors of human players in table soccer, a game recorder, KiRe, is constructed. The construction of KiRe is challenging because of the space limitation of a normal game table. A frame is designed using a CAD/CAM software, so that the turning of a game rod can be measured by a magnetic turning encoder, and the movement is estimated by an optical distance sensor. The position of the ball is measured by two synchronized Laser Measurement Systems (LMSs). These LMSs are set behind the

goals. The recorded data are accurate and with a very high frame rate. Similar mechanical units are designed and used in Star-Kick.

In KiRe, tracking the game ball is implemented using SAL, which can be described as a multiple Kalman Filters (KFs) approach. There are four learners. First, the training data are generated from the ball model. Then, a decision tree is built from the training data. Next, the thresholds of the segmentation are computed. Finally, the belief factors of the KFs are recomputed. The tracking is successful in the experiments. Based on these achievements, we develop an automatic referee for table soccer.

The recorded game data is a sequence of the measurements of the sensors, which are several real numbers mapped to the rotation and the position of the rods and the ball. Conditional Random Fields (CRFs) are employed to bridge the high-level skills of the human player to the low-level sensor data. CRFs is a discriminative probability model for explaining sequential data, whereby the most possible explanation of the sequential sensor data can be computed.

A layered-structure of CRFs is implemented within the SAL framework. The features in CRFs can be induced by using an incremental approach. We define a reduction step, whereby the number of features can be reduced while maintaining the performance of the CRFs. Moreover, a queue of CRFs can be learned and work together for the prediction. Their performance is guaranteed to be no worse than the single CRFs. In SAL, the learners are iterated in the following order: features are induced; features are reduced; CRFs are trained; a queue of CRFs are built.

The CRFs method mentioned above is proven to work well in a simulation. To explain the real data of table soccer, a set of the labeled data is required as the training set. We showed in the simulation that the methods are successful because annotating such a training set of real table soccer data is too tedious to be finished. In the experiments with the real data, we show that the methods work, but the results are not useful because of the limited amount of the labeled data.

The second game – Tetris, has been one of the most popular computer games for over ten years. The game can be found in most desktop platforms, e.g. KDE, gnome, and Windows. The Tetris game in KDE: KBlocks, is chosen as the basis of the platform for the research. Several new features are developed in KBlocks. The game engine, graphic user interface, and the players can be connected via Ethernet, into which the different players, AI or human, can easily be plugged and engage in a competition.

Using learning by imitation, we developed a method to learn several Support Vector Machines (SVMs) for playing the Tetris game. Instead of search-

ing for the highest "rewards" during the learning process, learning by imitation is driven by searching for the "similarities". There is a layered-structure in SVMs, implemented within the SAL framework, being similar to the situation of the CRFs mentioned above. The features of SVMs are a number of patterns. These patterns, as well as a series of hand-coded features, are used as the features in SVMs. Seven SVMs are learned for playing the games. The experiments show that the performance of the player can be significantly improved when our player acquires similar game skills to those of the imitated humans. Our player can play Tetris in diverse ways by imitating different players, and has chances to defeat the best-known artificial player in the world. The framework supports incremental learning because the artificial player can find stronger players and imitate their skills.

We also develop an interesting game controller which is not related to SAL. Rather than being easy and comfortable, this game controller is designed to be physically taxing for the players. It consists of several sensors, which makes a game more lively and forces the users to be more physically active. By using different mapping methods, one game can be played in several ways. The statistics gathered from the experiments show that even though the quality of control on the chosen fighting game is not as high as with a normal joystick, the developed controller is still preferred by most of the participants. It induces much more movement than a normal joystick.

In summary, SAL is defined and studied in the games, table soccer and Tetris. The platforms are developed, in which SAL can be applied. The similar layered-structure can be found in the methods used in both of the games. The experiments show that SAL works fine in these applications.

16

Discussion

Many systems have a layered-structure. For instance, if the language of human beings is regarded as a system, these layers can be easily found from one perspective: the words or vocabulary is the lowest layer, which defines single and isolated meaning. Phrases, e.g. noun phrases, consist of several words. The phrases can be regarded as the joining of the words. A sentence, normally consisting of several phrases, can express an ideas. Several sentences form a paragraph. The paragraph can be used to describe strategies or high-level skills.

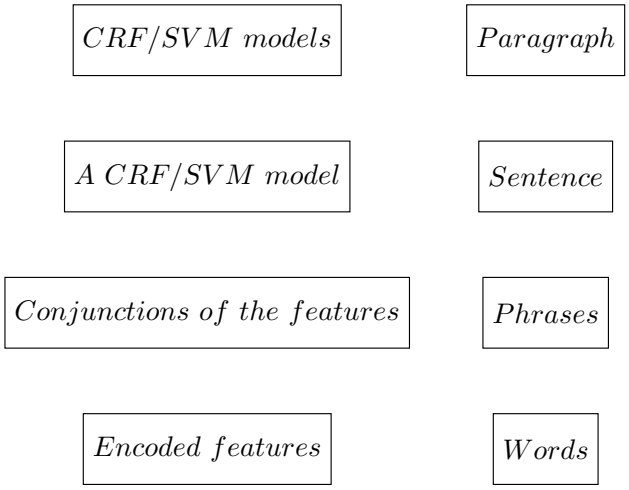


Fig. 16.1. The Layered Structure of the Systems

Figure 16.1 shows the layered structure of the language, as well as those of the CRFs and SVMs. There are four layers in the structure. From bottom to top, a higher layer is based on the contents of the lower one. In the first layer, basic features are defined. These features are normally hand-coded. In the second layer, the algorithm choose the features and conjoins them together. In the third layer, these conjunctions are used in the training of the models, e.g. CRFs or SVMs. In the fourth layer, the trained model is added into the known model set. After the forth layer, the set of the models should be employed to filter the training data, so that the filtered data can be used again to induce features in the first layer. In other words, a new round from the first layer can be started.

Table soccer robots, as the application, can already win even against advanced human players in the games. In our research, we did not develop towards playing the games even stronger, but use it as the test-bed of SAL. The training of CRFs, which is employed for explaining the sequential data, requires an annotated data set. This set can hardly be manually obtained. We thus use the simulation in the experiments. For the next steps, the plan would be exploring semi-supervised learning, in which both the labeled and the unlabeled data can be utilized for better performance. After that, making the robots imitate and learn from the better players, and developing the artificial player to be the best player would be the ultimate goal of our research.

Regarding the work on CRFs, we define a feature reduction step and CRF queue. The experiments are done in the simulation. CRFs is a research-oriented area, in which acceptable results or inventions should be tested in the widely acceptable benchmarks. We did not do this work because our research is application-oriented. The experiment on the benchmarks in natural language processing would not help to improve the performances of the agent in the games. However, we should test the feature reduction step and CRF queue on the benchmarks from the research point of view.

A method using learning by imitation is developed for playing Tetris. In the experiments, our player has a chance to win against the hand-coded player, which is regarded as a benchmark. Although the similarity is searched in our method, our player cannot play exactly the same game given the sequence of the blocks of the imitated player. This dissimilarity indicates that the features, which are at the lowest layer, do not have enough power of expression. Some of the ideas of the imitated system cannot be expressed by the combination of the features and SVMs. For the next step, we will try SAL in simpler games easier than Tetris. We hope that this simplicity can offer a good view on how SAL works in the game.

Human beings, as the vessel of intelligence, can use introspection and correct their faults during the practice. SAL provides a way to do similar things. One learner can be improved by the others.

References

[1] M.J. Matarić A. Billard. Human arm movement by imitation:evaluation of biologically inspired connectionist architecture. *Robotics and Autonomous Systems*, 35:145–160, 1998.

[2] Z. Sahel A. Bouchachia, B. Gabrys. Overview of some incremental learning algorithms. pages 1–6, 2007. In Proceedings of Fuzzy Systems Conference, 2007. FUZZ-IEEE.

[3] I. Szita A. Lőrincz. Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18:2936–2941, 2006.

[4] P. Auer and J. Kivinen. Finite-time analysis of the multiarmed bandit problem. In *Machine Learning*, pages 235–256, 2002.

[5] J. K. Baker. The Dragon System – An Overview. *IEEE Trans. Acoust. Speech Signal Processing*, ASSP-23:24–29, 1975.

[6] A. Bandura. Social learning theory. *New York: General Learning Press*, 1971.

[7] L.E. Baum and T. Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *Ann. Math. Stat.*, 37:1554–1563, 1966.

[8] P. R. Bélanger, P. Dobrovolny, A. Helmy, and X. Zhang. Estimation of angular velocity and acceleration from shaft-encoder measurements. *The International Journal of Robotics Research*, 17(11), November 1998.

[9] D. C. Bentivegna, A. Ude, C. G. Atkeson, and G. Cheng. Learning to act from observation and practice. *International Journal of Humanoid Robotics*, 1, December 2004.

[10] H. J. Berliner, G. Goetsch, M. Campbell, and C. Ebeling. Measuring the performance potential of Chess programs. volume 43, pages 7–20, 1990.

[11] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Workshop on Computational Learning Theory (COLT)*, 1998.

[12] J. Bohn. The smart jigsaw puzzle assistant: Using rfid technology for building augmented real-world games. In *Workshop on Gaming Applications in Pervasive Computing Environments at Pervasive 2004, Vienna, Austria*, apr 2004.

[13] C. Breazeal and B. Scassellati. Robots that imitate humans. *Trends in Cognitive Science*, 6, 2002.

[14] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[15] B. Brügmann. Monte-Carlo go, 1993. Unpublished technical report.

[16] M. Buro. From simple features to sophisticated evaluation functions. In *Computers and Games, Proceedings of CG98, LNCS 1558*, pages 126–145. Springer-Verlag, 1999.

[17] B. Scassellatib C. Breazeala. Robots that imitate humans. *Trends in Cognitive Sciences*, 6-11:481–487, 2002.

[18] D. Carr. Adapting reinforcement learning to tetris. 2005. Bachelor Thesis of Rhodes University, Grahamstown 6139, South Africa.

[19] S. Schaal C.G. Atkeson. Learning from demonstration. pages 12–20, 1997. In Proceedings of 14th International Conference on Machine Learning (ICML97).

[20] C. C. Chang and C. J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[21] O. Chapelle, V. Sindhwani, and S. S. Keerthi. Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, 9:203–233, June 2008.

[22] M. Chen, Y. Chen, M.R. Brent, and A.E. Tenney. Gradient-based feature selection for conditional random fields and its applications in computational genetics. In *Proceedings of Twenty-First IEEE International Conference on Tools with Artificial Intelligence (ICTAI09)*, 2009.

[23] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[24] A. Craig, B. Leonard, D. Edward, H. Rowell, J. James, L. Daniel, M. Neil, and Wartellaellen. Voilent video game effects on children and adolescents. 2003.

[25] A. Hornung D. Zhang, B. Nebel. Switching attention learning - a paradigm for introspection and incremental learning. pages 99–104,

2008. In Proceedings of Fifth International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2008).

[26] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):pp. 1–38, 1977.

[27] G.T. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML04)*, 2004.

[28] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classier under zero-one loss. pages 103–130, 1997.

[29] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of International Conference on Machine Learning*, 1995.

[30] K. Driessens. Relational reinforcement learning. 2004. PhD thesis, Catholic University of Leuven.

[31] C. Fehey. Tetris AI. http://www.colinfahey.com/tetris/tetris_en.html, 2003. www accessed on 02-August-2010.

[32] S. Fei and P. Fernando. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL03)*, volume 1, pages 134–141, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[33] M. Fox, M. Ghallab, G. Infantes, and D. Long. Robot introspection using learned hidden markov models. *Artificial Intelligence*, 170:59–113, 2006.

[34] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. pages 119–139, 1997.

[35] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classiers. pages 131–163, 1997.

[36] S. Greenberg and C. Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 209–218. ACM Press, 2001.

[37] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. 3:1157–1182, 2003.

[38] P. Hämäläinen and J. Höysniemi. A computer vision and hearing based user interface for a computer game for children. In *Proceedings of 7th ERCIM Workshop, "User Interfaces for All"*, Oct. 2002.

[39] R. Headon and R. Curwen. Movement awareness for ubiquitous game control. *Personal Ubiquitous Comput.*, 6(5-6):407–415, 2002.

[40] H. J. Herik, J. W.H.M. Uiterwijk, and J. Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134:277–311, 2002.

[41] H. Ishii and B. Ullmer. Tangible bits: towards seamless interfaces between people, bits and atoms. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, New York, NY, USA, 1997. ACM Press.

[42] M. Kearns and L. G. Valiant. Learning boolearn formulae or finite automata is as hard as factoring. Technical report, Harvard University, Aiken Computation Laboratory, August 1995.

[43] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, pages 282–293. Springer, 2006.

[44] D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A Generalized Hidden Markov Model for the Recognition of Human Genes in DNA. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, pages 134–142, Menlo Park, CA, June 1996. AAAI Press.

[45] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of 18th International Conference on Machine Learning (ICML 2001)*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[46] Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, and Tomoichi Takahashi, editors. *RoboCup 2006: Robot Soccer World Cup X*. Springer, 2007.

[47] B. Leskes and L. Torenvliet. The value of agreement a new boosting algorithm. *Journal of Computer and System Sciences*, 74:557–586, June 2008.

[48] L. Liao, T. Choudhury, D. Fox, and H. Kautz. Training conditional random fields via gradient tree boosting. In *Proceedings of Twentieth International Joint Conference on Artificial Intelligence (IJCAI07)*, 2007.

[49] F. V. Lishout, G. Chaslot, and J. W.H.M. Uiterwijk. Monte-Carlo tree search in Backgammon, 2007.

[50] R. Dillmann M. Pardowitz, R. Zöllner. Learning sequential constraints of tasks from user demonstrations. In *Proc. IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS), 2005*, Dec 31 2005.

[51] M. J. Mataric. Getting humanoids to move and imitate. *IEEE Intelligent Systems*, 15(4):18–24, 2000.

[52] M. Mateas. Expressive ai: Games and artificial intelligence. In *In Proceedings of Level Up: Digital Games Research Conference*, pages 322–335, Utrecht, Netherlands, 2003.

[53] A. McCallum. Efficiently inducing features of conditional random fields. In *Proceedings of 19th Conference in Uncertainty in Articifical Intelligence (UAI03)*, 2003.

[54] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[55] G. Kókai S. Mandl N. Böhm. An evolutionary approach to tetris. 2005. In Proceedings of the sixth metaheuristics international conference (MIC2005).

[56] B. Nebel, T. Weigel, and J. Koschikowski. Tischfussball, hockey oder dergleichen und verfahren zur automatischen ansteuerung der an stangen angeordneten spielfiguren eines tischspielgeräts für fussball-, hockey- oder dergleichen, 2005.

[57] J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society*, University of California, Irvine, Aug. 1985.

[58] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. pages 267–296, 1990.

[59] L. Reyzin. 2 player tetris is pspace hard. 2006. In Proceedings of 16th Fall Workshop on Computational and Combinatorial Geometry.

[60] T. Ryden, T. Teräsvirta, and S. Asbrink. Stylized facts of daily return series and the hidden markov model. *Journal of Applied Econometrics*, 13(3):217–244, 1998.

[61] A. L. Samuel. Some studies in machine learning using the game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[62] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3, 1999.

[63] J. Schaeffer and R. Lake. Solving the game of Checkers. In *Games of No Chance*, pages 119–136. Cambridge University Press, 1996.

[64] R. E. Schapire. The boosting approach to machine learning: An overview. 2003.

[65] R.E. Schapire. The strength of weak learnability. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:28–33, 1989.

[66] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423,623–656, 1948.

[67] H. Shatkay and L. P. Kaelbling. Learning topological maps with weak local odometric information. In *Proceedings of the Fifteenth Interna-*

tional Joint Conference on Artificial Intelligence (IJCAI), pages 920–929, 1997.

[68] V. Sindhwani and D. S. Rosenberg. An rkhs for multiview learning and manifold co-regularization. In *Proceedings of the 25th Annual International Conference on Machine Learning, (ICML08)*, pages 976–983, 2008.

[69] M. Suwa. A Congnitive Model of Acquiring Embodied Expertise Through Meta-cognitive Verbalization. In *Proceedings of 1st International Symposium on Skill Science (ISSS07)*, pages 80–86, Keio University, Tokyo, Japan, 2007.

[70] S. Bruno T. Christophe. Building Controllers for Tetris. *International Computer Games Association Journal*, 32:3–11, 2009.

[71] M. Tacke, T. Weigel, and B. Nebel. Decision-theoretic planning for playing table soccer. In T. Frühwirth S. Biundo and G. Palm, editors, *Advances in Artificial Intelligence. Proceedings of the 27th Annual German Conference on Artificial Intelligence (KI04)*, pages 213–225. Springer-Verlag, 2004.

[72] J. Tang. Ai agent for tetris. 2009. Bachelor Thesis, University of Freiburg, Germany.

[73] T. Toyoda and Y. Kotani. Monte Carlo Go using previous simulation results. *Technologies and Applications of Artificial Intelligence*, 0:182–186, 2010.

[74] L. G. Valiant. A theory of the learnable. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 436–445, New York, NY, USA, 1984. ACM.

[75] N. Villar, K. M. Gilleade, D. Ramdunyellis, and H. Gellersen. The voodooio gaming kit: a real-time adaptable gaming controller. *Comput. Entertain.*, 5(3):7, 2007.

[76] R. S. Virginia. Learning classification with unlabeled data. In *Neural Information Processing Systems (NIPS93)*, pages 112–119, 1993.

[77] S. Vishwanathan, N. Schraudolph, M. Schmidt, and K. Murphy. Accelerated training of conditional random fields with stochastic meta-descent. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML06)*, 2006.

[78] T. Weigel. Kiro – a table soccer robot ready for the market. *Künstliche Intelligenz*, 01/05, 2005.

[79] T. Weigel and B. Nebel. Kiro – An Autonomous Table Soccer Player. In *Proceedings of RoboCup Symposium '02*, pages 119 – 127, Fukuoka, Japan, 2002.

[80] T. Weigel and B. Nebel. Tischfussball: Mensch versus computer. 31:323–332, 2008.

[81] T. Weigel, K. Rechert, and B. Nebel. Behavior recognition and opponent modeling for adaptive table soccer playing. In U. Furbach, editor, *Advances in Artificial Intelligence. Proceedings of the 28th Annual German Conference on Artificial Intelligence (KI05)*, pages 335–350. Springer-Verlag, 2005.

[82] T. Weigel, D. Zhang, K. Rechert, and B. Nebel. Adaptive vision for playing table soccer. In T. Frühwirth S. Biundo and G. Palm, editors, *Advances in Artificial Intelligence. Proceedings of the 27th Annual German Conference on Artificial Intelligence (KI04)*, pages 424–438. Springer-Verlag, 2004.

[83] M. Weiser. The computer for the 21st century. *Scientific American*, 02/1991 1991.

[84] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995.

[85] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, 1995.

[86] D. Zhang. Action selection and action control for playing table soccer using markov decision processes. Master's thesis, Albert-Ludwigs-Universität, Freiburg, Germany, 2005.

[87] D. Zhang, Z. Cai, and B. Nebel. Playing Tetris using learning by imitation. In *GAMEON*, pages 23–27. EUROSIS, 2010.

[88] D. Zhang and A. Hornung. A table soccer game recorder. In *Video Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS08)*, Nice, France, 2008.

[89] D. Zhang and B. Nebel. Learning a table soccer robot a new action sequence by observing and imitating. In *Proceedings of the Third Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE07)*, pages 61–67, 2007.

[90] H. Zhang. The optimality of naive bayes. In V. Barr and Z. Markov, editors, *Proceedings of FLAIRS Conference*. AAAI Press, 2004.

[91] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th Annual International Conference on Machine Learning, (ICML03)*, pages 912–919, 2003.

[92] X. Zhu and A. B. Goldberg. *Introduction to Semi-Supervised Learning*. Morgan and Claypool, 2009.